JUL 15 '93 15:48

PAGE.003

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE OCT 28, 1993	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
MANAGING THE TRANSITION TO OBJECT-ORIENTED TECHNOLOGY FOR DEPARTMENT OF DEFENSE INFORMATION MANAGEMENT SYSTEMS

5. FUNDING NUMBERS

6. AUTHOR(S)
MR. DAVID H. DISKIN, DISA/JIEO/CFSW;
MS. KATHLEEN A. JORDAN, IDA; MR RICHARD P. MORTON, IDA;
MR. ROBERT FURICK, IDA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Defense Information Systems Agency
JIEO/Center for Software, Code TXE
701 South Courthouse Road
Arlington, VA 22204-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Same as above.

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Field 25 should contain the identifier CIM (Collection), as detailed in A. Washington DTIC-OCS IOM, dated April 11, 1994.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
Distribution is unlimited.

12b. DISTRIBUTION CODE

19950123 090

RACT (Maximum 200 words) This document presents the findings and recommendations for managing the implementation of object-oriented technology (OOT) in the DoD within specific information technology areas. This document explores some of the technology transition issues related to OOT implementation and identifies alternative strategies for such a transition within the DoD. The approach to this task was to consider the implementation of OOT with regard to current and planned DoD policy, current and anticipated DoD needs as expressed by DoD representatives, and the general state of maturity of OOT theory and practice. A key premise of this effort is that OOT has matured to the state whereby it may be considered for use within the DoD information technology areas. OOT appears to offer significant advantages over process and data-driven software development approaches, such as increased maintainability and reusability. OOT also facilitates distributed computing. However, there still exists a diversity of concepts and notations for development methodologies and a variety of mechanisms for object-oriented database technology. There appear to be relatively few, if any, impediments within DoD information system lifecycle management and software development standards to using OOT.

14. SUBJECT TERMS
object oriented design, object oriented analysis, Ada, Ada9x, information management

15. NUMBER OF PAGES
61

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

MANAGING THE TRANSITION TO OBJECT-ORIENTED TECHNOLOGY FOR DEPARTMENT OF DEFENSE INFORMATION MANAGEMENT SYSTEMS

October 28, 1993

Prepared by

David H. Diskin

DEFENSE INFORMATION SYSTEMS AGENCY
Joint Interoperability Engineering Organization
Center For Information Management
Software Systems Engineering Directorate
701 S. Courthouse Road, Arlington, Virginia 22204-2199

Supported by

Kathleen A. Jordan, Task Leader
Robert Furick
Richard P. Morton
INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

ABSTRACT

This document presents the findings and recommendations for managing the implementation of object-oriented technology (OOT) in the DoD within specific information technology areas. This document explores some of the technology transition issues related to OOT implementation and identifies alternative strategies for such a transition within the DoD. The approach to this task was to consider the implementation of OOT with regard to current and planned DoD policy, current and anticipated DoD needs as expressed by DoD representatives, and the general state of maturity of OOT theory and practice. A key premise of this effort is that OOT has matured to the state whereby it may be considered for use within the DoD information technology areas. OOT appears to offer significant advantages over process and data-driven software development approaches, such as increased maintainability and reusability. OOT also facilitates distributed computing. However, there still exists a diversity of concepts and notations for development methodologies and a variety of mechanisms for object-oriented database technology. There appear to be relatively few, if any, impediments within DoD information system lifecycle management and software development standards to using OOT.

PREFACE

This document was prepared by the Defense Information Systems Agency, Joint Interoperability Engineering Organization, Center for Information Management, sponsored by the Office of the Director of Defense Information, now the Office of the Deputy Assistant Secretary of Defense (Information Management). The work was supported by the Institute for Defense Analyses (IDA).

The following people served as reviewers of this document:

Mr. Andrew Baer	American Management Systems, Inc.
Mr. Brad Balfour	SofTech, Inc.
MSgt Ted Blanchard	USAF, Standard Systems Center
Mr. Grady Booch	Rational
Mr. Dave Bulman	Pragmatics, Inc.
Mr. Dave Carter	DISA/JIEO/Center for Information Management
Col Randy Catts	Ballistic Missile Defense Office
Ms. Sherrie Chubin	DISA/JIEO/Center for Information Management
Dr. Brad Cox	George Mason University
Ms. Corinne Engle	DISA/JIEO/Center for Integration & Interoperability
Dr. Edward Feustel	Institute for Defense Analyses
Dr. Michael Frame	Institute for Defense Analyses
Dr. Fred Hathorn	Office of the Deputy Assistant Secretary of Defense (Information Management)
Ms. Deborah Heystek	Institute for Defense Analyses
Mr. Kenneth Kelley	DISA/JIEO/Center for Information Management
Mr. David Kriegman	Systems Research & Applications Corp.
Mr. Huet Landry	DISA/JIEO/Center for Standards
Dr. Robert Marcus	Boeing Computer Services
Dr. Reginald N. Meeson	Institute for Defense Analyses
Dr. Steven Merritt	DISA/JIEO/Center for Information Management
Ms. Tamra Moore	DISA/JIEO/Center for Information Management
Mr. Don Reifer	Institute for Defense Analyses

Mr. Gerard Russomano	DISA/JIEO/Center for Information Management
Dr. Randy Scott	Software Productivity Consortium
Mr. Robert Shelton	Open Engineering, Inc.
Mr. Ben Stivers	Harris Corp.
Mr. Chris Stone	Object Management Group
Mr. Glen White	Institute for Defense Analyses

EXECUTIVE SUMMARY

PURPOSE

The purpose of this document is to present the findings and recommendations of an initial study regarding the implementation of object-oriented technology (OOT) for information management systems in the Department of Defense (DoD). This task was conducted by the Defense Information Systems Agency (DISA), Joint Interoperability Engineering Organization, Center for Information Management (CIM), Software Systems Engineering Directorate, supported by the Institute for Defense Analyses (IDA). The Office of the Deputy Assistant Secretary of Defense for Information Management directed DISA to investigate the potential advantages and disadvantages of applying OOT to the areas of software development methodologies, software development standards, lifecycle process models, the DoD Technical Architecture Framework for Information Management (TAFIM), the Integrated Computer-Aided Software Engineering (I-CASE) Program, the CIM Process Model for Information Management, the DoD Data Administration Program, the DISA/CIM Software Reuse Program, and the DISA/CIM Metrics Program. This document also explores some of the technology transition issues related to OOT implementation and identifies strategies for such a transition.

BACKGROUND

This report is primarily concerned with the application of OOT for automated information systems (AISs), systems that are data intensive and transaction processing oriented. To date, most of these systems have been written in Cobol, using either process-driven or data-driven analysis and design techniques. In addition, the average size of these systems is approximately 250,000 lines of code, with the largest approaching 3 million lines of code. There is a limited amount of new development, with most work in software maintenance. These systems have been developed under a variety of process standards and some without a structured lifecycle process at all.

APPROACH

The approach to this task was to consider the implementation of OOT with regard to current and planned DoD policy, current and anticipated DoD needs as expressed by DoD representatives, and the general state of maturity of OOT (theory and practice). The information was gathered through a review of both academic and trade literature; meetings with several DISA organizations; meetings of DISA representatives in an ad hoc OOT working group; attending the February 1993 Object World Conference in Boston and the April 1993 Object Expo Conference in New York; interviews with software developers from American Management Systems Inc., Systems Research and Applications Inc., NASA/Goddard Space Flight Center Software Engineering Laboratory; and a video teleconference and site visit to the Standard Systems Center at Gunter Air Force Base, Alabama.

AREAS ADDRESSED

The Object Paradigm in Software Engineering

The object paradigm represents a shift in our perspective regarding software. At its essence, this paradigm represents another mechanism for managing complexity whether of the software code, design specification, or the real-world problem to be solved. The features of encapsulation and inheritance support increased maintainability and reusability which leads to lower costs in software development and maintenance. Disadvantages of the object paradigm include the diversity in its concepts and notations and the relative immaturity of object-oriented data management technology.

Impact on Selected DoD Policies, Programs, and Standards

This effort examined the use of OOT in conjunction with the following selected DoD policies, programs, and standards:

- **Lifecycle Management of Automated Information Systems:** The current lifecycle management policy should accommodate object-oriented technology since the policy provides an incremental and evolutionary lifecycle model which should be suited to object-oriented development.
- **Software Development Standards:** The new draft DOD-STD-SDD will allow the specification of "objects" and otherwise does not appear to present any methodological or structural impediments to using OOT.

- **I-CASE Program:** The Minimum Features for the I-CASE Methodology Support do not require an object-oriented methodology, but instead the requirement is established as a Tier 1 Feature, meaning that it is expected to be commercially available in the near term. The CIM Process Model for Information Management (IM), which is part of the I-CASE Request for Proposal, requires an IDEF (Integrated Computer-Aided Manufacturing DEFinition) approach for the functional process improvement activity which precedes system development. If system development uses an object-oriented approach, there will be a shift in paradigms from an IDEF perspective to an object-oriented perspective, and there is no straight-forward conversion from one perspective to another.
- **TAFIM Data Management:** Database support for OOT is still immature. Today, ODBMS (Object-Oriented Database Management System) technology is best suited for applications involving complex data structures where retrieval requires navigating on the basis of relationships in the data; ODBMS technology does not yet appear appropriate for traditional transaction processing applications. For now, object-oriented systems should use relational database technology for simple business data. Additionally, most ODBMS bindings are only available for C++ or Smalltalk, with only one binding under development for Ada9X.
- **TAFIM Programming Services:** In the current version of the TRM, the only identified language standard is Ada83, which is "object based," not "object oriented." When the new Ada9X becomes DoD policy, the TRM will specify a true object-oriented programming language. In the meantime, however, languages such as C++ and Smalltalk, designed specifically to support OOT, do so more completely than Ada83. Most class libraries are available in C++ and Smalltalk, with only recent class library development in Ada9X.
- **TAFIM Distributed Computing:** Although OOT appears to facilitate distributed computing through the use of encapsulation, the integration of OOT and distributed computing technology is so immature that the very nature of the relationship is still a matter for debate. Experience with distributed computing in general and object-oriented distributed computing in particular is very limited.

- **DISA/CIM Reuse Program:** The existing repository can store and retrieve object-oriented code and analysis, design, and prototype models, but the current faceted scheme offers a limited search mechanism when compared to object/class libraries and browsers that are commercially available. The DSRS does allow a user to see those components that have a defined relationship with a specific component. It should be noted that this capability is provided by the DSRS software and not the faceted classification scheme. For relationships between components to be identified, they must be specified by the repository librarian when the component is entered into the library. The search capability is limited in that the DSRS does not allow a search based upon a specific relationship a component may have with other components.
- **DoD Data Administration:** Primitive data elements (object attributes) can be managed with the current Data Administration procedures in the DoD. The notion of standard data does not need to change to accommodate OOT. If a data administration program were to take a pure object-oriented approach, then instead of managing data models and data elements (attributes), it would manage object/classes with their associated attributes and operations. This repository of object/classes would look very similar to a reuse repository of object/class definitions. Standard data elements could be viewed as domain-independent objects that have concrete external representations and only a minimal set of generic operations (methods).
- **DISA/CIM Metrics Program:** Traditional metrics may need to either be replaced or reinterpreted. Experience with OOT metrics is still quite limited and much more work is needed, especially for project estimation and design quality and complexity. As a measure of size, the DISA/CIM metrics program uses lines of code, which most OOT practitioners believe is inappropriate for OO based software development.

Technology Transition

The recommended strategy will focus at the DoD enterprise level on removing impediments to OOT adoption and at the organization level on providing guidance to plan OOT insertion. OOT must be transitioned to an organization with regard to its existing environment, the training and experience of personnel, the type of systems to be built, and the anticipated opportunities for software reuse from future projects. From an analysis of a number of case studies and technology transition literature, some basic "lessons learned" and transition strategies have been identified.

Lessons Learned:

- The most significant lesson from the technology transition case studies is that OOT is not a substitute for software engineering principles, and the promised benefits of OOT will happen only if sound management and engineering practices are in place.
- The "right" design and analysis methodology is not a settled issue.
- Identifying and designing classes, particularly reusable classes, requires problem domain expertise, not just methodology expertise.
- The benefits from reuse may not be realized until several object-oriented projects have been completed and reuse libraries have been built.
- Both the technology transition literature and the case studies noted that there will be a significant learning curve, with developers needing at least six months to one year to become proficient.

Transition Strategies: Based upon the experience papers and case studies, there appear to be three basic strategies to transitioning to object-oriented technology: (1) system/project based, (2) domain based, and (3) enterprise based. In the first case, OOT is adopted as the result of the needs of a specific system development, not as a corporate-wide adoption. In the domain-based strategy, OOT is used across multiple systems and projects but within the same domain. In the third case, OOT is adopted enterprise-wide as part of a corporate strategy for system development.

CONCLUSIONS

In considering whether the DoD should implement OOT within its information system development, we have reached the following conclusions:

OOT appears to offer advantages over traditional process-driven and data-driven approaches. The features of encapsulation and inheritance support increased maintainability and reusability which lead to lower costs in software development and maintenance. OOT also appears to facilitate distributed computing through the use of encapsulation which encourages a client-server perspective.

OOT is not a completely mature technology. There still exists a diversity of concepts and notations for development methodologies and a variety of mechanisms for an object-oriented database technology. For now, relational database technology should remain the primary method for simple business data, with object database technology more appropriate for complex data types.

There appear to be relatively few, if any, impediments within DoD AIS lifecycle management and software development standards to using OOT. The existing DoD lifecycle process model for AISs and the anticipated software development standard should accommodate the use of OOT.

There are impediments and technical risks within the DoD Process Model for Information Management (IM) and the TRM that should be resolved prior to any large-scale transition to OOT. The Process Model for IM may need to be revised to resolve the IDEF to object-oriented transition between functional process improvement and system development. The TRM may need to be revised to accommodate object-oriented database technology.

A successful transition to object technology will require tools and a significant training effort. Since, the success of OOT will depend in large measure upon the capability of the individuals who use it, training in object concepts along with software engineering concepts is essential, as is the support of automated tools.

RECOMMENDATIONS

The overall recommendation of this effort is that the DoD should support the introduction of OOT into its software engineering and information technology strategy. Its use should be considered as another tool within a larger toolkit of software engineering mechanisms and approaches. The introduction of this technology will need to consider functional process improvement activities, software development processes and standards, legacy systems, and emerging technical and functional architectures. A specific organizational move to OOT should consider a number of factors: the maturity of the technology, the ability of the organization to accept and use it, and the economic justification. The following are some specific recommendations and actions regarding the implementation of OOT.

Develop a technology transition strategy for the introduction of OOT. A technology transition strategy will establish priorities for transition activities, and define the scope and stages for the transition, the roles and responsibilities, and the mechanisms necessary for a transition to occur. This strategy should meet the mission and information technology needs and should consider the existing base of people and systems and the anticipated opportunities for software reuse.

Reevaluate the Process Model for IM. The model should be reevaluated to resolve the shift in paradigms between the Functional Process Improvement activity, which uses an IDEF approach, and the system development activity, which could potentially use an object-oriented development approach.

Do not mandate OOT as an approach for all software development nor adopt any specific object-oriented methodology or technique at this time. Further analysis is needed to determine which object-oriented technique or techniques are desirable for a particular application development. Software engineering activities require a variety of techniques, including event and process modeling and prototyping. Considerations such as performance and efficiency could conceivably require a non-object-oriented design.

Make OOT concepts part of Ada and software engineering training. Develop and institute a training program that incorporates OOT in software engineering, Ada, and design and analysis methods.

Support efforts toward a commonly accepted definition for OOT. A definition is needed that accommodates Ada and the software engineering concepts that support high reliability and maintainability. This will have an effect upon the mechanisms used for defining objects, particularly comparing the inheritance vs. encapsulation qualities of OOT.

Investigate the feasibility of extending the current DSRS to more fully support object/class search and retrieval. The current faceted scheme offers a limited search mechanism when compared to object/class libraries and browsers that are commercially available.

Encourage the development of ODBMS bindings for Ada9X. Currently, most ODBMS bindings exist for either the C++ or Smalltalk programming languages. The DoD should encourage the development of ODBMS bindings for Ada9X, including the investigation of emerging standards such as SQL3 and ODMG-93 (Object Data Management Group) specifications.

Develop a standard set of Ada class libraries. All commercial object-oriented languages contain a set of library classes shipped as an intrinsic addition to the language standard. These classes are usually built-in abstractions such as sets, collections, arrays, bags, and strings. These built-in classes have standard methods associated with them. To aid the development of object-oriented applications in Ada9X, the DoD should develop a standard set of library classes similar to the predefined classes in Smalltalk and C++.

Support ongoing and new research in OOT metrics. The DoD should launch an initiative to identify a core set of practical OO metrics, comparable to those defined by the SEI for traditional development.

TABLE OF CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

1.1 PURPOSE AND SCOPE

The purpose of this document is to present the findings and recommendations of an initial study regarding the implementation of object-oriented technology (OOT) for information management systems in the Department of Defense (DoD). This task was conducted by the Defense Information Systems Agency (DISA), Joint Interoperability Engineering Organization, Center for Information Management (CIM), Software Systems Engineering Directorate, supported by the Institute for Defense Analyses (IDA). In response to this new technology, the Office of the Deputy Assistant Secretary of Defense for Information Management (ODASD(IM)) directed the DISA to investigate the potential advantages and disadvantages of applying OOT.

This document provides a preliminary assessment and recommendations regarding the implementation of OOT within the information technology areas of requirements analysis, software design, functional process improvement, software development standards, lifecycle process models, the DoD Technical Architecture Framework for Information Management (TAFIM), the Integrated Computer-Aided Software Engineering (I-CASE) Program, the CIM Process Model for Information Management, the DoD Data Administration Program, the DISA/CIM Software Reuse Program, and the DISA/CIM Metrics Program. This document also explores some of the technology transition issues related to OOT implementation and identifies lessons learned and strategies for such a transition.

1.2 BACKGROUND

As software systems have become more complex, the software engineering community has employed a number of different approaches for structuring software specifications and code. These approaches have been used in an attempt to make the software more maintainable, thus reducing overall lifecycle costs. Until recently, most approaches separated functions and data, structuring software around the functions to be performed and organizing data around information flows. Object-oriented approaches, however, address both functions and data together, structuring software specifications and code around objects rather than functions. Object-oriented approaches to software analysis, design, and programming have become increasingly popular, and the object-oriented perspective has also been extended to other areas of information technology such as database design and software reuse.

The DoD is exploring new information technologies such as OOT in an effort to produce high quality, reliable, and maintainable software while also reducing software development and maintenance costs. In its draft *Software Technology Strategy* [DOD91], the DoD expressed the following objectives for its software and information technology base for the year 2000:

- Reduce equivalent software lifecycle costs by a factor of two.
- Reduce software problem rates by a factor of 10.
- Achieve new levels of DoD mission capability and interoperability via software.

In a similar manner, the objectives of DoD's Corporate Information Management Initiative are to reduce information system development and maintenance costs and to "deliver DoD growing requirements with less resources." The DDI has a goal of an "80/20% development/maintenance ratio," with a "technology asset life that would be two to three times greater than the technology innovation cycle" [STR91]. The DoD has employed the so-called "traditional" technologies such as process and data-driven approaches to software development. And while the approach alone is not the sole determinant for software cost efficiency and quality, it does have a major effect upon those aspects. As a result, the DoD is investigating OOT as another means to improve software quality and to reduce software costs.

As the CIM effort is primarily concerned with automated information systems (AISs), the application of OOT must be appropriate for systems that are data intensive and transaction processing oriented. To date, most of these systems have been written in Cobol, using either process-driven or data-driven analysis and design techniques. In addition, the average size of these systems is approximately 250,000 lines of code, with the largest approaching 3 million lines of code. There is a limited amount of new development, with most work in software maintenance. These systems have been developed under a variety of process standards and some without a structured lifecycle process at all. As a result, for many systems, there are few requirements and design artifacts. Current estimates for

Central Design Activity (CDA) staffing range from 33,000 to 34,000 computer specialists and 3,400 to 3,500 computer scientists. Their background is primarily in Cobol, with about one-half having attended college [DIS93].

1.3 APPROACH

The approach to this task was to consider the implementation of OOT with regard to current and planned DoD policy, current and anticipated DoD needs as expressed by DISA representatives, and the general state of maturity of OOT (theory and practice). This task considered the general areas of information system development and maintenance, including requirements analysis and specification, design, reuse, architectures, databases, data administration, and technology transition.

The information gathering was accomplished through a review of both academic and trade literature; meetings with several DISA organizations; meetings of DISA representatives in an ad hoc OOT working group; attending the February 1993 Object World Conference in Boston and the April 1993 Object Expo Conference in New York; interviews with software developers from American Management Systems Inc., Systems Research and Applications Inc., National Aeronautics Space Administration (NASA)/ Goddard Space Flight Center Software Engineering Laboratory; and a video teleconference and site visit to the Standard Systems Center at Gunter Air Force Base, Alabama.

1.4 ORGANIZATION OF DOCUMENT

Section 2 discusses the object-oriented paradigm in software engineering, reviewing the object definition, its advantages and disadvantages, its effect in analysis and design methodologies, and its implications for reengineering. Section 3 considers the impact of OOT upon selected DoD policies, programs, and standards, including DoDD 8120.1, DoD-STD-SDD, the TAFIM, the I-CASE effort, the DoD DISA/CIM Reuse program, the DoD Data Administration program, and the DoD DISA/CIM Metrics program. Section 4 discusses the lessons learned and general strategies for transitioning to OOT. Section 5 summarizes the conclusions and provides specific recommendations for the implementation of OOT with regard to the areas noted above. Appendix A reviews each of the case studies.

2. THE OBJECT PARADIGM IN SOFTWARE ENGINEERING

At the heart of OOT is the object paradigm, and while the object paradigm itself is not new, its application to information technologies has grown significantly within the last five years. The object-oriented paradigm represents a shift in our perspective regarding software. At its essence, this paradigm represents another mechanism for "managing complexity" [BOO87] of the software code, design specification, and the real-world problem to be solved. Originally applied to programming languages, the object paradigm has been applied to software development methodologies, database technology, and software reuse. This section examines the changes implied by the object paradigm in software engineering, with specific attention to analysis and design methodologies and the implications for the reengineering of legacy systems. This section also considers the general advantages and disadvantages of the object paradigm when compared to process-driven and data-driven software engineering approaches. Issues related to business process and system lifecycle models, databases, and programming languages are discussed in Section 3 in the context of specific DoD policies, programs, and standards.

2.1 DEFINITIONS

If we consider object-oriented languages and software development methodologies as instances of the object paradigm, we would find that there is still variability in concept and notation [SNY93, NEL91]. This may be due to the fact that the object paradigm is not formalized; and possibly because object technology can be applied to different activities within the software lifecycle. Wegner [WEG90] notes that the programming language used may affect the object-oriented paradigm. The Object Management Group (OMG), a non-profit industry consortium, now has a standard set of terminology and concepts for the object paradigm. But beyond OMG's efforts, there appears to be relatively little public technical discourse regarding object definitions. There are occasional articles about the object definition; however, these analyses generally do not look at the programming, design, and analysis perspective together. See Snyder [SNY93] for further discussion regarding the diversity of definitions.

Below are a set of definitions taken from several sources, and where there exists some notable diversity, more than one definition has been included. These definitions are not a complete list, but are included here to explain some of the more common ideas behind OOT.

Object: *From a programming language perspective*, an object is a "self-contained set of variables which can only be manipulated by a set of methods (procedures) defined exclusively for that purpose" [NEL91].

From an analysis and design perspective, an object is "something you can do things to. An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class. The terms instance and object are interchangeable" [BOO91]; . . . "an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both; an encapsulation of Attribute values and their exclusive Services" [COA90]. From a general perspective, "All objects embody an abstraction. An object explicitly embodies an abstraction that is meaningful to its clients. Although an object may involve data, an object is not just data" [SNY93].

Class: "A class can be defined as a description of similar objects, like a template or cookie cutter" [NEL91]. The class of an object is the definition or description of those attributes and behaviors of interest.

Operation/Method: "Some action that one object performs upon another in order to elicit a reaction [BOO91]." "A Service is a specific behavior that an Object is responsible for exhibiting" [COA90].

Message: Mechanism by which objects request services of each other. Sometimes this is used as a synonym for operation.

Abstraction: "Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental properties" [RUM91].

Encapsulation: "(also information hiding) consists of separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects" [RUM91]. "The act of grouping into a single object both data and the operations that affect that data" [WIR90].

Inheritance: "Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship" [RUM91]. "Subclasses of a class inherit the operations of their parent class and may add new operations and new instance variables. Inheritance allows us to reuse the behavior of a class in the definition of new classes" [WEG90].

Polymorphism: The same operation may behave differently on different classes [RUM91]. Subclasses may have different forms (implementations), yet all conform to a common profile defined by a superclass.

There is general agreement within the object-oriented community regarding the basic object-oriented principles of encapsulation, inheritance, and polymorphism. However, work remains to be done in formalizing the object model and specific implementation issues arising from these principles. Wegner [WEG90] provides a relatively thorough discussion of the object paradigm. However, he believes that the formalism may vary based upon the object-oriented language considered. Wegner notes the existence of "subparadigms" of object-based programming: object based, class based, and object oriented. He continues that these "subparadigms . . . associated with Ada, Smalltalk, and Actors likewise determine nonoverlapping research communities that rarely talk to each other. Small linguistic differences yield large paradigm shifts" [WEG90]. At this time, most formal models of object-orientation appear to be based on the object-oriented languages that implement them.

2.2 OBJECT-ORIENTED LANGUAGES

Although the object paradigm has recently become popular, it is not a new concept. In programming languages, the concept of OOT was introduced over 25 years ago with the development of Simula in 1967 by Ole Dahl et al. [DAH70]. Simula introduced the idea of classes and objects, which is considered central to the object notion. Smalltalk, which was developed in the 1970s by the Xerox Palo Alto Research Center Learning Research Group, extended the object/class notion by making everything in the language an object. Smalltalk evolved throughout the 1970s with releases every two years, culminating in Smalltalk-80. Along with this were the ideas of "layers of abstraction" [DIJ69], information hiding [PAR72], and abstract data types [LIS74]. The languages CLU and Alphard provided some of the first implementations of abstract data types and encapsulation, although without inheritance. By the late 1970s, these ideas were coming under the scope of what we now call "object oriented." Object-oriented concepts of inheritance and encapsulation were incorporated into non-object-oriented languages such as C resulting in C++ and into Pascal resulting in Ada and Object Pascal [BOO91]. By the 1980s the object-oriented paradigm began to accommodate software design, initially as a support for software engineering in Ada [FIC92a]. It was later extended for requirements analysis building upon ideas in entity-relationship data modeling.

In general, languages that support all object-oriented principles (abstraction, encapsulation, inheritance, and dynamic polymorphism), such as Smalltalk, are considered "object oriented"; languages that support only abstraction, encapsulation, and static polymorphism, such as the current version of the Ada programming language, Ada83, are considered "object based" [BOO91]. The anticipated revision to Ada, Ada9X will have mechanisms for true single inheritance and dynamic polymorphism, making it a true object-oriented language.

2.3 SOFTWARE DEVELOPMENT METHODOLOGIES

In addition to object-oriented programming languages, several object-oriented software development methodologies have been created over the last 10 years. These methodologies generally include elements to support both object-oriented analysis (OOA) and object-oriented design (OOD). These methodologies present an alternative to the structured analysis (process-driven) and information engineering (data-driven) methodologies traditionally employed with information system development.

Compared to object-oriented programming languages, object-oriented software development methodologies take a different perspective regarding objects. Object-oriented development methodologies view an object as something in the real world to be modeled and represented in software. The objects in analysis and design are those perceived in the "real world" (problem domain) such as persons, tangible things, and events [COA90] and those in the "system" (solution domain) such as scroll bars, menus, and windows. The abstractions for analysis and design are based upon those "objects."

One of the earliest OOD methodologies was developed by Booch [BOO83] as an accompanying design technique with the Ada programming language [FIC92]. As such, this technique was specific to the Ada language but has since been expanded and refined to accommodate other programming languages. In addition to Booch's methodology (which has been updated [BOO91]), other methodologies have appeared, with varying degrees of analysis and design elements, most of these within the last five years. These methodologies include those developed by [BUH90, MEY88, WIR90, RUM91, COA90, SHL92, JAC92, EMB92, and MAR92].

Object-oriented methodologies provide a series of techniques and notations for modeling software. The primary model is the "object" model, which is a model of objects/ classes with their associated attributes and operations. Next, most methodologies create a dynamic model which defines the behavior of each object. This is accomplished with a state-based technique such as state transition diagrams. Last, a functional or process model is created for the identified states of the object. In general, most object-oriented methodologies consist of three basic views: object, dynamic, and functional.

With regard to object-oriented concepts, there appear to be two general approaches to these development methodologies. Coad, Rumbaugh, Shlaer-Mellor, and Martin take an "extended data modeling" or "entity-driven" approach. These methodologies stress the definition of the class with its attributes, operations, and relationships to other classes. There is less emphasis upon the interaction and message passing between classes and more emphasis upon commonalities between classes and the construction of inheritance hierarchies and aggregation structures.

Jacobson, Booch, Colbert, and Wirfs-Brock emphasize object responsibilities and interactions. These approaches are more "specification oriented" or "responsibility driven" in that they emphasize the contract (i.e., interface) between interacting objects. Fichman [FIC92a] notes that some techniques emphasize inheritance, while others emphasize encapsulation and object-interaction, and that "the design trade-offs between maximizing encapsulation (by emphasizing object responsibilities) versus maximizing inheritance (by emphasizing commonalities among classes) are subtle ones."

2.4 COMPARISON TO PROCESS AND DATA-DRIVEN APPROACHES

The object-oriented perspective of software development presents a contrast to process-driven and data-driven approaches. In a process-driven (functionally oriented) approach, such as Modern Structured Analysis [YOU89], a domain is initially modeled as cooperating asynchronous processes (using DFDs). The problem domain is then modeled with entity-relationship diagrams using the data identified in the DFDs as a starting point. Next, the behavior is modeled using state-transition diagrams. The goal of a process driven approach is to produce a top-down functional decomposition of the intended software [FIC92a]. In this approach, three viewpoints are established: functional, entity (object), and dynamic, with the functional abstraction as the primary basis for domain partitioning. The design stage continues with further functional decomposition for the software implementation. In the object-oriented development methods, we find the same general views; however, the object/class abstraction is the primary basis for partitioning the domain, with the functional and dynamic views subordinate.

In a data-driven approach, such as information engineering [MAR92], the problem domain is first modeled by the entities observed, then by a variety of methods which include process decomposition and process dependency modeling. The design stage then consists of a functional decomposition of system functionality, with logic and state-based modeling until the development of algorithms in the pseudo-code referred to as action diagrams. The actions diagrams are then translated into source code. The data models are further refined for database design. Although entities are initially defined during the analysis phase, they do not serve as the basis for software design. As with the process-oriented approaches, the "process" serves as the basis for software design.

In comparing the object-oriented methodologies to traditional approaches, there are multiple views (object, dynamic, function) in each; however, in the object-oriented approaches, the object serves as the primary abstraction for both analysis and design, with behaviors and functions subordinate to the object. Fichman [FIC92a] observes that OOA methodologies represent a radical change from the functionally oriented structured analysis approaches but only an incremental change from the data-driven information engineering approaches. OOD methodologies represent a radical change from both process and data-driven design approaches.

With process-driven methodologies, the process serves as the primary abstraction for both analysis and design. In data-driven methodologies, the entity serves as the primary abstraction for analysis, but the process/function is the primary abstraction for design. In both the process and data-driven methodologies, the resulting software implementation is process/function oriented, with data globally available and not localized to specific procedures. Non-localized data leads to what is known as the "ripple effect" in software maintenance, in which changes to one module generate a "ripple" of changes in other modules. Thus, the maintenance problems associated with non-localized, global data are not solved with either the process or data-driven approaches.

2.5 IMPLICATIONS OF OBJECT-ORIENTED TECHNOLOGY ON REENGINEERING

Currently, a substantial portion of the DoD software development is expected to be the maintenance or reengineering of existing legacy systems. Legacy systems, particularly those that have evolved over many years, are often characterized by unstructured code, obsolete or non-existent documentation, and a mix of languages and platforms. Reengineering, which is "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" [CHI90], is motivated by a number of conditions: the need to upgrade its functionality or to improve its performance, or to move to another platform. A reengineering effort often includes reverse engineering the existing system to extract design and analysis information, refining the new specifications, and then forward engineering the new system based upon the new specifications. The reengineering may also require the refinement of existing code into a more structured form before it is reverse-engineered. The reengineered system could simply reflect a new design or could also incorporate new requirements.

Given that the overall design will probably be the result of some form of a functional decomposition process, functionally based analysis and design specifications will have to be "transformed" to an object-oriented model. Since this capability does not exist with current CASE tool technology, this transformation is still a manual process.

An approach to reengineering is the use of system "wrapping." This is where a system or part of a system is isolated within a specially constructed interface or wrapper. This interface provides an object-oriented appearance to other systems. Taylor [TAY92], who provides a discussion on system wrapping, notes that the design and implementation of the legacy system will affect the degree of wrapping that can occur. For example, systems that used a functional decomposition approach and have high inter-module coupling may be harder to break apart into "wrappable" pieces. Although system wrapping is not full-scale reengineering, it may offer an interim solution to transitioning legacy systems to an object-oriented environment.

If a non-object-oriented system is to be reengineered, the transition to OOT should be evaluated with a cost benefit analysis that considers the analysis and design approaches used, the extent of the reengineering effort, the potential for reuse (in other systems or domains), the anticipated life of the system, and any anticipated upgrades.

2.6 ASSESSMENT OF THE OBJECT PARADIGM

The object paradigm has some distinct advantages and disadvantages. Some are inherent in its concept and others are the result of its maturity. Although the object paradigm is still evolving, these features also have to be considered in comparison to the process and data-oriented methodologies, which at this time constitute the primary alternative paradigms for software development.

2.6.1 Maintainability

One of the most powerful features of OOT is encapsulation. This is essentially the capability to separate an abstraction (idea) from its implementation. In this manner, there is a specification which provides an external view and an implementation which provides an internal view. Objects can only "see" other object specifications and cannot make changes to the internal state of an object directly (for example, the direct assignment of an object's instance variable). In limiting the visibility between software components, the use of encapsulation can reduce the degree of coupling, thus making the software more maintainable and understandable. Software objects interact with each other through an interface specification. With this separation, it is possible to have the underlying implementation change without affecting the specification. Without a change to the interface, there should not be an effect upon other objects that use that interface.

A second aspect of OOT is that data and the associated operations for that data are localized within the software implementation. That is, data (which maintains the state of the object) is not globally visible to the entire system and can only be changed by those operations within its scope. This "informational cohesion" mitigates the "ripple effect" seen with functionally decomposed designs, where changes to data (which is globally visible) affect modules throughout the system. This feature of OOT makes it superior to process and data-driven approaches. With those approaches, data tends to be global with visibility unrestricted across the entire application [BOO87].

The use of inheritance, while very powerful, can also lead to very complex inheritance hierarchies. This complexity can be significant when multiple inheritance is used, resulting in possible inheritance clashes. Inheritance can also violate the principle of encapsulation by providing access to "hidden" inherited properties [NIE89]. Fichman [FIC92a] notes that the approach for defining objects using encapsulation (i.e., composition) is significantly different than that used with inheritance (i.e., classification).

Wilde and Matthews [WIL93] observe that the use of OOD (and OOT in general) may not necessarily lead to more maintainable systems if the object-oriented strategies lead to "a profusion of relatively small program parts with many potentially complex relationships." A developer attempting to derive a subclass from a library "must often look at method source code to see the other objects involved in the more complex algorithms. When they did need to read the code, they often had to trace long sequences of message sends to find [the subclass]" [WIL93].

2.6.2 Understandability

To some degree, the object notion is an attractive basis for modeling a system since we can easily observe "objects" in the real world. This ability serves the concern noted in the Ada language standard to consider software development as a "human activity" [DOD83]. In addition to programmer understandability, there appears to be improved communication with customers concerning requirements. Anecdotal evidence indicates that objects are easier to describe and discuss with customers than software functions and procedures.

2.6.3 Reliability

Where reliability is defined as the ability to detect and handle anomalous conditions, the reliability aspects of OOT will depend substantially upon the programming language chosen. For instance, Smalltalk is a weakly typed language (all variables are just "objects"), and methods are not matched against the object until run time. Ada and C++ are both strongly typed languages, so the methods and constraints of a variable are matched against the variable's type at compile time. Program reliability is enhanced with the use of strong typing since more errors will be detected at compile time rather than at run time.

2.6.4 Efficiency

There can be some performance risks associated with using OOT. Booch [BOO91] notes the performance risk related to the encumbrance of classes which results in excessive object code. Efficiency in object-oriented languages can be degraded if the language allows for dynamic polymorphism, that is, where methods are resolved (matched to objects) at run time. This resolution requires a search up the inheritance tree of classes to find the class that has the needed method. A dynamic method invocation may take 1.75 to 2.5 times as long as a simple subprogram call, but this overhead can be offset by limiting the use of run-time resolution [BOO91]. There are techniques to improve the lookup time such as method caching, in which searches are done with a single hash table lookup [RUM91].

2.6.5 Reusability

The object-oriented feature of inheritance can contribute substantially to software reuse. A new subclass can be defined in terms of an existing (parent) superclass, and new attributes and operations can be added to the subclass. NASA/Goddard's Software

Engineering Laboratory (SEL) believes that the use of OOD has also contributed to higher levels of reuse. OOD was used by the SEL in conjunction with Ada and domain analysis, with limited use of inheritance. In terms of reusability and reconfigurability of software, the SEL considered OOT the "most influential technology studied by the SEL" [STA93a].

One of the strengths of the object-oriented approach is the possibility for development of reusable class libraries or frameworks for specific domains. These libraries could be standardized across technical areas, or business domains, to improve productivity and quality. In the past two years, a number of these libraries have started to appear in certain technical areas. About a dozen commercial class libraries have been developed in specific technical areas, such as image processing or sound control for CD-ROM. As experience with OOT matures, commercial class products may develop in business domains such as hospital administration, logistics, or other business functions.

2.6.6 Consistency

The definition of the object paradigm itself varies to some degree in both concept and representation. These differences are especially noticeable when moving from the analysis and design perspective to the implementation perspective. Snyder [SNY93] states that "the groups involved with object technology lack a shared understanding of the basic concepts and a common vocabulary for discussing them." For example, he observes that an "object" is also known as an "instance, class instance, surrogate, [and] entity" and that the concept of encapsulation can mean three distinct concepts: "information hiding," "embedding," and "protecting." As noted earlier, the OMG now has a standard set of terminology and concepts for the object paradigm.

While most methodologies employ some form of a graphical notation to describe classes and relationships, there is no consensus yet regarding object-oriented notation. This may be one reason that CASE tool support for object-oriented development methods tends to be limited to the vendor providing the methodology. Mazzucchelli [MAZ92] observes that the developers of methodologies are also vendors of their own proprietary CASE tools. This differs from the structured methodologies market in which there are several CASE tools available for a single methodology. It should be noted that this situation may recently be improving: a couple of the more well-known methodologies such as Schlaer-Mellor and Rumbaugh are now supported by more than one tool vendor.

2.6.7 Empirical Evidence

Another weakness of OOT is that the bulk of empirical evidence showing that it is a superior software technology is just now emerging [LEI93]. Even though OOT is not a new technology, there are relatively few published case studies or experience papers with statistics on productivity. Of the 7 years of proceedings from the ACM Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA), there were 253 papers published. Of those, only Lewis et al. [LEW91] and Champeaux [CHA92] provided statistics on productivity or reuse improvements. Reuse and maintenance issues were usually addressed in the panel sessions held (there were 47 panels held during those years). Within the last year, there have been some published OOT success stories. However, most of these cases have been published by promoters of OOT. David Taylor [TAY92] reviewed 18 case studies where object technology has been implemented, and the OMG will soon publish a collection of 26 OOT success stories, called *Objects in Action*. Another possible reason for few published case studies is that OOT has been seen as a competitive edge by companies using it, and as a result companies have been unwilling to publicize their experiences [BAE93]. The most promising measures of experience come from the NASA/Goddard SEL, which has been maintaining statistics for over 16 years [STA93a].

3. IMPACT ON SELECTED DOD POLICIES, PROGRAMS, AND STANDARDS

One of the first concerns with introducing any new technology is whether the existing infrastructure of policy, procedures, standards, and organizations will support its implementation. This section examines the use of OOT in conjunction with the following selected DoD policies, programs, and standards:

- "Life-Cycle Management (LCM) of Automated Information Systems (AISs)" - DODD 8120.1 and DODI 8120.2
- Software Development and Documentation Standards
- Integrated Computer-Aided Software Engineering (I-CASE) Program
- Technical Architecture Framework for Information Management/Technical Reference Model (TAFIM/TRM)
- DISA/CIM Reuse Program
- DoD Data Administration Program
- DISA/CIM Metrics Program

3.1 LIFE-CYCLE MANAGEMENT (LCM) OF AUTOMATED INFORMATION SYSTEMS (AISs) - DODD 8120.1 and DODI 8120.2

Current policy regarding lifecycle process models is expressed in the DoD Directive (DoDD) 8120.1, "Lifecycle Management Phases and Milestones for Automated Information Systems" and its accompanying DoD Instruction (DODI) 8120.2, "Automated Information System (AIS) Life-Cycle Management (LCM) Process, Review, and Milestone Approval Procedures." DoDI 8120.2 defines four major types of program strategies for the development of an automated information system:

- **Grand Design Program Strategies:** "They are characterized by acquisition, development, and deployment of the total functional capability in a single increment. The required functional capability can be clearly defined and further enhancement is not foreseen to be necessary. A grand design program strategy is most appropriate when the user requirements are well understood, supported by precedent, easily defined, and assessment of other considerations (e.g., risk, funding, schedule, size of program, or early realization of benefits) indicates that a phased approach is not required."

- **Incremental Program Strategies:** "They are generally characterized by acquisition, development, and deployment of functionality through a number of clearly defined system 'increment's' that stand on their own. The number, size, and phasing of the 'increment's' required for satisfaction of the total scope of the stated user requirement must be defined by the AIS PM [Program Manager], in consultation with the functional user. An incremental program strategy is most appropriate when the user requirements are well understood and easily defined, but assessment of other considerations. . . indicates a phased approach is more prudent or beneficial."
- **Evolutionary Program Strategies:** "They are generally characterized by the design, development, and deployment of a preliminary capability that includes provisions for the evolutionary addition of future functionality and changes, as requirements are further defined. . . The total functional requirements the AIS is to meet are successively refined through feedback from previous increments and reflected in subsequent increments. Evolutionary program strategies are particularly suited to situations where, although the general scope of the program is known and a basic core of user functional characteristics can be defined, detailed system or functional requirements are difficult to articulate . . ."
- **Other Program Strategies**: "They are intended to encompass variations and/or combinations of the [above] program strategies . . ., or other program strategies not listed above; e.g., OMB Circular A-109 . . . acquisitions, commercial-off-the-shelf (COTS), nondevelopmental item (NDI), and commercial item acquisitions."

The 8120 directive and instruction are not the same as software development standards, such as DOD-STD-2167A or the draft DOD-STD-SDD. Those development standards define the required *products* (software and documentation) of the software development process. The 8120 instruction defines four basic *processes* that may be applied when developing software. The draft DOD-STD-SDD notes three (grand design, incremental, and evolutionary) options for software development processes, but does not require a specific type of process.

Support for OOT

Overall, the available OOT literature on lifecycle process models notes the need for an iterative process such as an incremental, evolutionary, and rapid prototyping approach for object-oriented software development [PIT93]. The iterative approach to software development has also become widely accepted with the recognition that is it very difficult to specify requirements completely and correctly before the design and implementation activity, and second, that even if requirements were specified "perfectly," they will very likely change throughout the life of the system. An iterative process allows the refinement of object models of requirements and design after they have been initially defined. The "grand design," waterfall approach is recognized as less desirable for software development in general, whether object oriented or not.

Object-oriented methodologies may vary in their support of an iterative lifecycle. Two software development projects by McDonnell Douglas [FAY91, FAY92] compared the use of two different OOA methodologies: one top-down and the other, bottom-up. This effort noted that the Shlaer-Mellor approach, which they considered bottom-up, was less amenable to an iterative approach due to the fact that all objects needed to be defined before other modeling (process and state) could continue. This effort also used Colbert's OOA approach, which was top-down in defining objects, and found that it worked better with an iterative lifecycle. The conclusion was that certain methodologies may be better suited to different lifecycle models. If the methodology works better with a "grand design," i.e., single phase development, then this type of effort should probably be one where the requirements are well understood at the beginning of development or perhaps a reengineering effort where no new requirements are being defined.

Assessment

The program strategies of DoDD 8120.1 and DODI 8120.2 can easily accommodate the use of object-oriented software development. The issue may be as to the type of methodologies chosen rather the allowable lifecycle models.

3.2 SOFTWARE DEVELOPMENT AND DOCUMENTATION STANDARDS

The current policy regarding software development and documentation is defined in two major standards: DOD-STD-2167A, "Defense System Software Development," and

DoD-STD-7935A, "DoD Automated Information Systems (AIS) Documentation Standards." These two standards are currently being harmonized into a single standard known as Software Development and Documentation (SDD) Standard, DOD-STD-SDD, the draft version, and when approved, DOD-STD-498. The latest draft of the SDD, dated 22 December 1992, has been issued for formal coordination and review. The SDD will combine the current standard for embedded weapons software development (DOD-STD-2167A), with the standard for automated information systems (DOD-STD-7935A). The SDD, which is now in its fifth revision, is expected to be approved in 1993.

Support for OOT

Within requirements analysis, where OOA would apply, there are no methodologies imposed directly by any of the standards. The Data Item Description (DID) for a Software Requirements Specification (SRS) of DOD-STD 2167A requires a specification of software functions with their associated inputs and outputs, and the Functional Description (FD) of DOD-STD 7935A is also a description of software "functionality."

The creation of the new standard should eliminate a number of problems perceived with the earlier standards. There is a specific objective in the SDD, to "improve compatibility with non-hierarchical design methods, e.g., object-oriented . . ." The SDD does not require a specific requirements or design methodology. Requirements and design specifications can be not only of the required functionality but of "objects," thus accommodating an object-oriented approach.

With regard to software development process, DOD-STD 2167A and DOD-STD 7935A did not mandate a particular process model. However, DOD-STD 2167A specified certain software development activities and milestones as a required part of the development. This specification of activities with milestones was often interpreted as a "grand design," waterfall process. The SDD recognizes three of the process models (grand design, incremental, and evolutionary) described in DoDD 8120.1/DODI 8120.2 as valid strategies for software development. The SDD also removes the rigid requirement for development milestones so that a waterfall process is not unintentionally imposed.

Assessment

Overall, the SDD does not appear to present any methodological or structural impediments to using object-oriented methodologies such as OOA or OOD. Nor does the standard require a software development lifecycle that would impede the development of object-oriented software models or components.

3.3 INTEGRATED COMPUTER-AIDED SOFTWARE ENGINEERING (I-CASE) PROGRAM

The I-CASE program will provide the DoD Central Design Activities with a software engineering environment using a combination of (COTS) hardware and software components and run-time licenses. This environment will provide full lifecycle, automated tool support for business case analysis, software development, maintenance, and reengineering of AIS applications. The implementation of the I-CASE environment will take place over the next three to five years, based upon the success of the pilot programs. When operational, the I-CASE effort will provide support for integration, system services, user services, hardware, software, maintenance, and training [ICA92].

Currently, the acquisition of the I-CASE environment is in the pre-award demonstration phase, with demonstrations being conducted at each offeror's site. Contract award is anticipated in late October or early November 1993, with a formal test and acceptance period through February 1994. The initial I-CASE pilot programs are expected to begin around May or June 1994.

The support for OOT provided by the anticipated I-CASE environment was evaluated by reviewing the I-CASE Request for Proposal (RFP) [ICA92] in the areas of software engineering methodologies, business case analysis, and the I-CASE Process Model for Information Management (IM). The I-CASE features detailed in the RFP are divided into three categories: Minimum, Tier 1, and Tier 2. Minimum Features are those that must be available at the time of contract award. Tier 1 Features are expected to be commercially available in the near term but are not required at contract award. Tier 2 Features are expected to be commercially available but not in the near term.

3.3.1 Software Engineering Methodologies

The I-CASE RFP specifies a variety of tools including those to support different software development methodologies. Tool support for any particular software development methodology is specified under "Methodology Support" (section 10.3.2.2.3).

Support for OOT

Within methodology support of the I-CASE RFP (section 10.3.2.2.3), the Minimum Features require that the I-CASE SEE (software engineering environment)

> provide software tools that support at least one specific, widely used, software development methodology that conforms to the life cycle process described in Section J, Attachment 4, Process Model for Information Management. The software development methodology shall be any or all of (1) functional; (2) data driven; (3) object-oriented; and (4) state transition oriented [ICA92, p.170].

Under the Tier 1 Features, the I-CASE SEE should

> provide software tools (e.g., combination of tools) to support object-oriented and one other development methodology outlined in paragraph 10.3.2.2.3.a.

Under Tier 2 Features, the I-CASE SEE should

> provide software tools to support three or more different software development methodologies . . ., one of which is object-oriented; support tailoring of each methodology; and be able to capture the output as assets for reuse [ICA92, p. 170].

Assessment

Since the Minimum Features for the I-CASE methodology support do not require an object-oriented methodology, this capability will depend upon the specific tool suites offered. Although it is likely that object-oriented methodology support will be a part of the proposed tool suites, there is no guarantee that an object-oriented tool will be supplied with the initial delivery. Additionally, if an object-oriented tool is proposed, it will to some degree impose its own variant of the object-oriented development methodologies with it.

3.3.2 Business Case Analysis

In addition to software methodology support, the I-CASE SEE will provide support for business case analysis. In business case analysis, which precedes software development, business models are developed to support a number of business assessment activities, including risk analysis, activity-based costing, and transform flow analysis.

Support for OOT

For business case analysis the I-CASE SEE will not support an object-oriented approach but rather an IDEF approach to modeling the enterprise. The RFP (section

10.3.2.3.2.2) states that the I-CASE SEE will "provide a capability to produce a high-level functionally-oriented business case analysis that supports the concept of continuous process modernization and quality improvement." Minimum features are to provide "business case analysis functionality (IDEF0 and IDEF1X)." The IDEF0 model is a process model representing the activities of an enterprise. The IDEF1X model is an information model which represents the "structure and semantics" of information in an enterprise.

Assessment

A point of concern, however, is the approach used for business case analysis. As noted, the business case analysis approach starts with a process model in IDEF0 and then develops an information model in IDEF1X. Although the entities in the IDEF1X are initially derived from the IDEF0 model, the correlations between process and data are not as easily localized as when taking an object-oriented approach. In an object-oriented approach, processes are subordinate to data. With the IDEF approach, the data are subordinate to processes in the initial IDEF0 modeling. Using both the IDEF0 and IDEF1X models, it is possible to construct an object model, but the transformation to an object model from the IDEF0 and IDEFIX models is a process that requires an interpretation of the models' semantics; approaches for transforming these models are still under development and are generally proprietary.

3.3.3 Process Model for Information Management (IM)

The I-CASE specification has incorporated the Process Model for IM (Figure 1) as its process for information management and systems development. The basic philosophy of the Process Model is that applications, data, and infrastructure have different lifecycles and hence should be developed and maintained separately. Previously, data and applications have been tied together in the same system, resulting in system-specific data elements which have little interoperability with other systems. In a similar manner, system-specific and proprietary hardware and software (infrastructure) have presented impediments to interoperability. These tightly coupled systems of data, applications, and infrastructure became vertical "stovepipes" that were not interoperable or compatible. The objective of the Process Model was to define a process with associated products that allowed for the separate lifecycles of data, applications, and infrastructure.

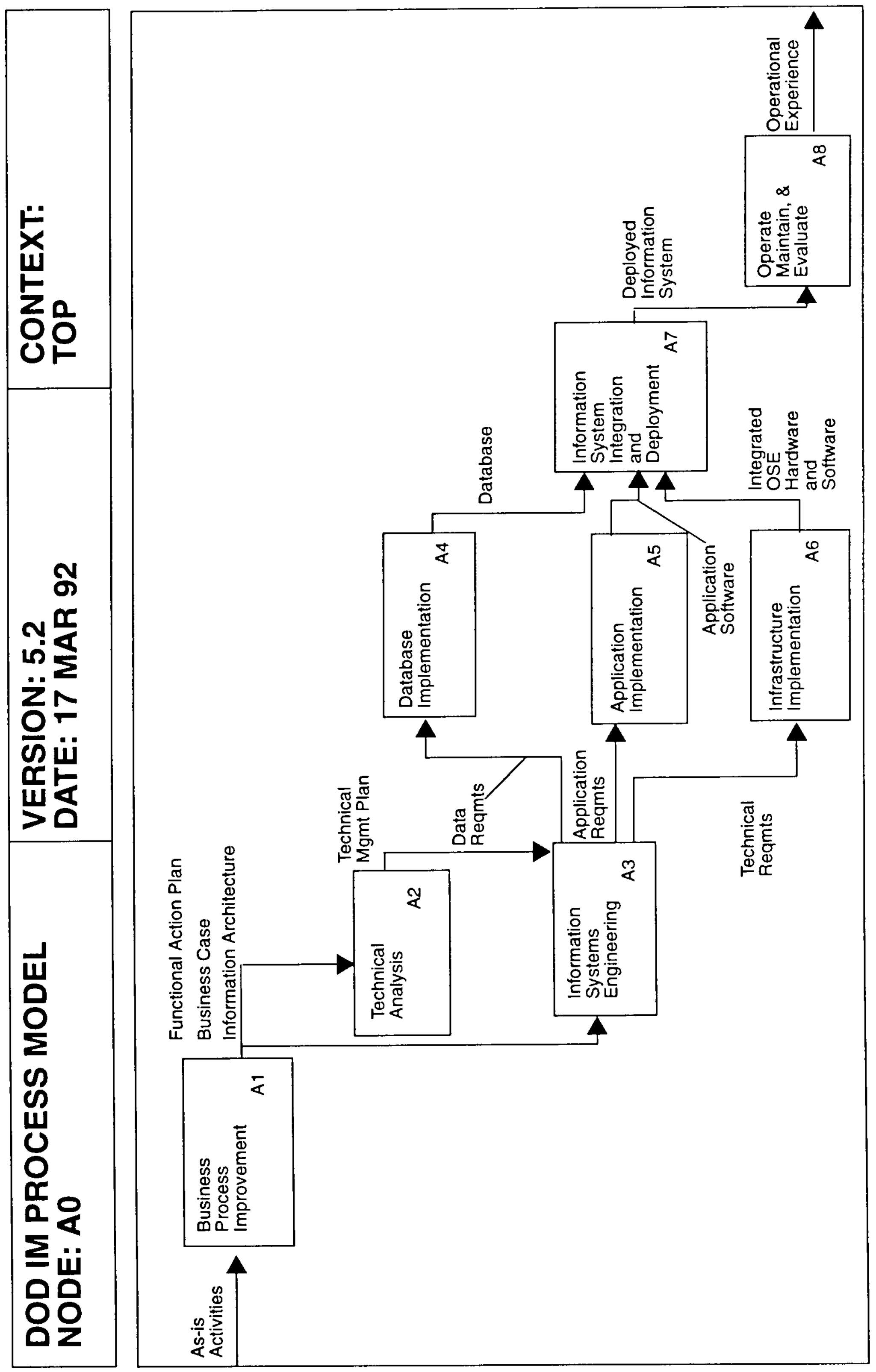

Figure 1. Process Model for IM

Support for OOT

There is little explicit mention of OOT in the Process Model. The model accommodates a variety of development approaches for application development, but uses an IDEF0 and IDEF1X approach for business process modeling. OOD is included in the Process Model as one of several mechanisms within the Design activity of Application Implementation. Otherwise, there is no specific mention of OOA methods or OOT for database development or infrastructure services. The data management portion of the Process Model is defined explicitly in terms of "data," "data architectures," and "data requirements," and not "objects."

Assessment

As noted earlier, the Functional Process Improvement (Business Case Analysis) activity uses IDEF0 and IDEF1X to model the business domain. If the system development, which follows Functional Process Improvement, uses an object-oriented approach, there will be a shift in paradigms from an IDEF0 and IDEF1X to object-oriented perspective. There is no straight-forward conversion from one perspective to another. An object-oriented approach to enterprise modeling should make the transition from Functional Process Improvement to system development easier, but it remains to be determined whether object-oriented enterprise modeling is appropriate for the functional domain.

3.4 TECHNICAL ARCHITECTURE

Existing architectural policy is provided in the form of the Technical Architecture Framework for Information Management (TAFIM). One particular volume of the TAFIM is the Technical Reference Model (TRM), which describes an open system environment in terms of an architectural model and a standards profile. This section addresses the impact OOT would have on the environment described in the TRM (see Figure 2). While there may be minor impact on all of the architectural components, the most significant impact will be on the following selected areas:

a. Data Management Services

b. Programming Services

c. Distributed Computing Services

In all cases, the impact is described here as being exclusively on the standards profile and not on the architectural model. This reflects the fact that the TRM is intended to describe the environment in which mission applications exist, and the services required to

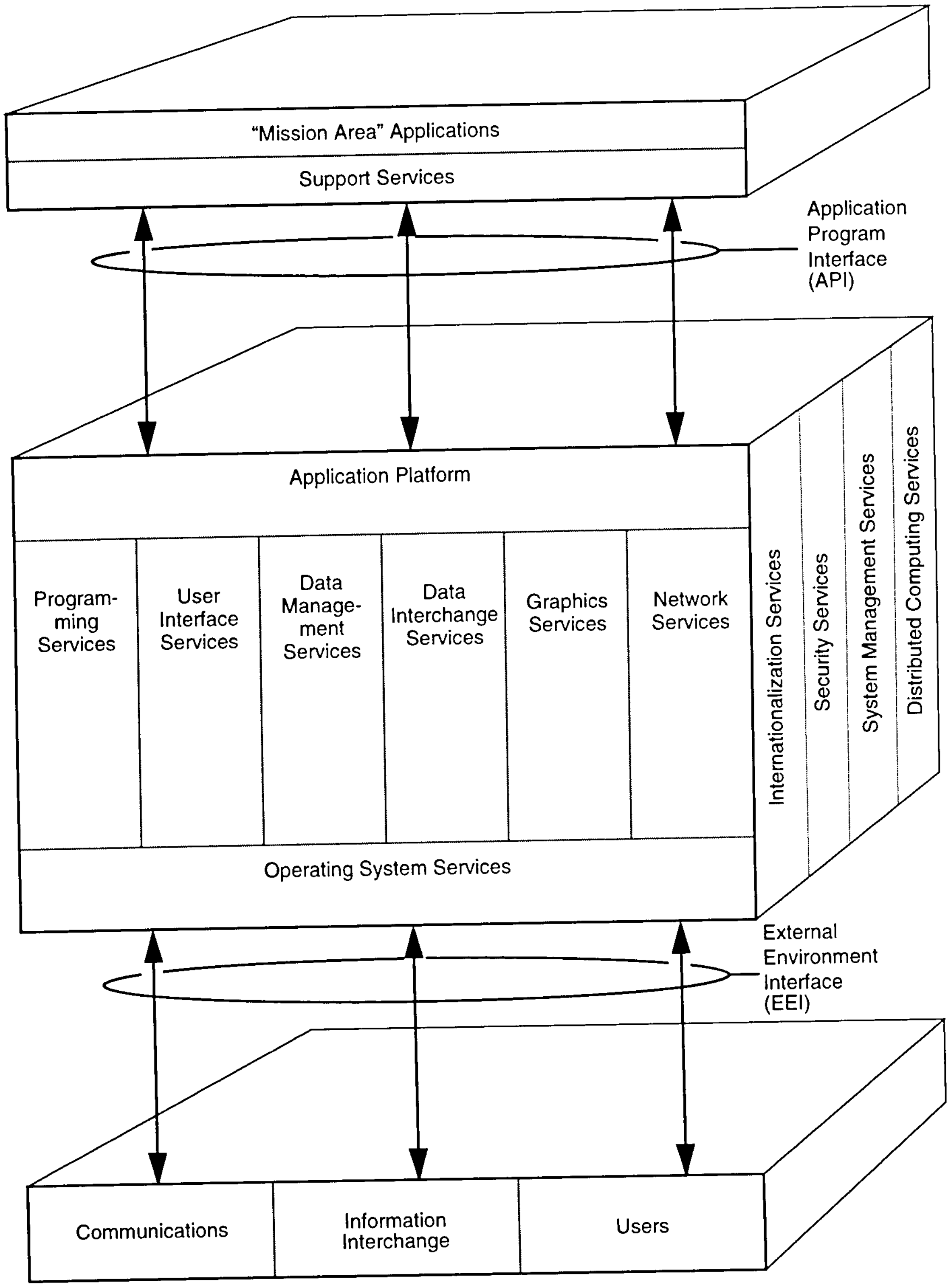
"Mission Area" Applications
Support Services
Application Program Interface (API)
Application Platform
Programming Services
User Interface Services
Data Management Services
Data Interchange Services
Graphics Services
Network Services
Internationalization Services
Security Services
System Management Services
Distributed Computing Services
Operating System Services
External Environment Interface (EEI)
Communications
Information Interchange
Users

Figure 2. DoD Technical Reference Model

support them. The actual grouping of services into service areas is somewhat arbitrary, reflecting the natural divisions that occur in the industry. These groupings do not have to change to support OOT. Certain services need to be added; these can easily be mapped into the existing service areas.

3.4.1 Data Management Services

The need for additional data management services relates to the need to store, update, and retrieve objects in much the same way that current systems store, update and retrieve data. An object-oriented database management system (ODBMS) is a system that provides a persistent store for the objects of an object-oriented environment along with the same types of services found in a traditional DBMS, such as backup and recovery, concurrency control, consistency control, access control, and query processing. In general, the primary motivations for using a DBMS of any kind are data sharing (which necessitates the control services) and query processing. Mere persistence can more easily be obtained through the use of files. The traditional DBMS concepts need to be adapted to object-oriented perspectives in some cases, particularly those related to query processing. An ODBMS is not the only way to provide DBMS services to an object-oriented environment. The rest of the system may be entirely object oriented with the DBMS being relational or some other type.

State of the Technology

The existing TRM only recognizes SQL as a standard interface to database management systems. SQL3, currently under development, is expected to support OOT. However, enthusiasts of ODBMS reject the notion of extending what is essentially relational technology as an approach to implementing an ODBMS.

Several vendors offer products that claim to be an ODBMS. However, there is no generally agreed model of object database management. As a result, at this time there is no agreement on what services an ODBMS should provide. A subgroup of OMG is attempting to define such a standard. However, vendors of relational systems are likely to back the SQL3 approach that embeds support for OOT within relational products. Still, principal sponsors of the OMG effort have agreed to produce conformant products within one year of the standard's publication.

The performance of ODBMSs appears to be superior to that of relational DBMS (RDBMS) for some applications and inferior for others. In systems that have to deal with large, complex objects, for example, systems used to design hardware and software, one vendor of ODBMS technology claims performance improvements of two to three orders of magnitude. But that same vendor admits that for large transaction processing applications, relational technology is still substantially better. Over time, the performance of ODBMS technology is expected to improve to make it more appropriate for a wider range of applications, but it is unlikely to ever replace other DBMS technology completely.

At the moment, there is no interoperability between competing ODBMS products. The promised OMG standard is intended to rectify that situation.

Most of the experience with ODBMSs is in areas of hardware and software design. In general, only small pilot projects have been attempted in business data processing.

Most existing ODBMSs have language bindings only for C and C++, with some support for Smalltalk. A binding for Ada is currently under development by EVB, Inc. for the Itasca ODBMS, with an expected release by November 1993.

Assessment

Database support for OOT is still immature. It appears that there is something, sometimes a lot, to be gained from optimizing a DBMS for particular types of access patterns, and the applications for which OOT has been the technology of choice seem to be one of the cases where relational technology is not most optimum. However, this may be an implementation issue more than a data model issue. As experience with object-based applications grows and their requirements for DBMS support become better understood, performance will undoubtedly improve. Adopting OOT will require DBMS support for objects when data sharing and query processing are needed, but it is not yet clear if the database management area of the architecture will have to be augmented to permit both relational and OOT technologies, or if support for OOT within an extended relational system will be adequate. In fact, as experience with OOT in general grows and the types of applications to which it is applied expands, some of them may still benefit more from traditional relational DBMS technology. Today, ODBMS technology is best suited for applications involving large complex objects where retrieval requires navigating on the basis of relationships in the data, but ODBMS technology is not yet appropriate for traditional transaction processing applications.

3.4.2 Programming Services

The area of Programming Services is defined to include the full range of support used in the development of engineered software. In every phase of the software engineering life cycle, if the work of that phase uses OOT, then the tools required to support that work will be different from those of the more traditional approaches.

Object-oriented programming languages date back to the late 1960s with the development of Simula. Shortly after that, Smalltalk was introduced and made available and extolled widely in the United States. Several other object-oriented languages were introduced more recently, most notably C++ which became very popular because it would run on virtually any platform that supported C.

State of the Technology

The most popular object-oriented programming language is C++, with Smalltalk getting most of the remaining marketshare. Both C++ and Smalltalk are supported by developers of tools and reusable object libraries. Commercial object-oriented components of other areas of the architecture, such as an ODBMS, generally provide bindings for one or both of these two languages and rarely any others.

The current version of the Ada programming language, Ada83, is considered object based, not object oriented, since it lacks constructs for inheritance and dynamic binding; it does, however, provide strong support for encapsulation with the package and private type constructs. A form of single inheritance is possible using derived types. The anticipated Ada9X will provide mechanisms for true single inheritance and dynamic polymorphism. With these features Ada9X will be considered an object-oriented language.

Assessment

The current version of the TRM defines the Programming Services area to include programming languages, bindings, and CASE tools and environments, but the only identified language standard is Ada, which is also Congressionally mandated. Since the current Ada83 is "object based," the current TRM does not specify a true object-oriented language. When the new Ada9X becomes DoD policy, that gap will be filled and there will be no issue with respect to the availability of an approved object-oriented programming language. In the meantime, however, languages designed specifically to support OOT do so more completely than Ada83. There is currently no policy related to the use of CASE tools and environments, but the pending procurement of I-CASE may result in a policy directive to require its use. For more information on I-CASE and OOT tools, see Section 3.3.

This, however, represents only the policy aspect of object-oriented programming services. The fact remains that most of the current work in object-oriented programming is being done in C++ or Smalltalk. It is primarily in these languages that commercially available class libraries and ODBMS bindings can be obtained. Class libraries and bindings for Ada have just recently become available with more under development.

3.4.3 Distributed Computing Services

Distributed computing services are those concerned with the mechanisms that permit service requestors to interact with service providers in a transparent way. Both requestors and providers may be applications or the implementations of platform services. The specification of just how much behavior is included within the domain of distributed computing has not been agreed to completely within the industry. Several options have been proposed under the label of a reference model or architecture for distributed computing. They are all treated as reference models in the discussion that follows. Evaluating the impact of the introduction of OOT on the distributed computing services depends on the selection of the distributed computing reference model (DCRM) made by DoD.

The specific definition of what services are included within distributed computing depends upon the choice of a DCRM. Several such models have been proposed, including the following:

a. Advanced Networked Systems Architecture (ANSA), a research product of the Alvey Program in the United Kingdom, and now managed by Architecture Projects Management Limited

b. Distributed Computing Environment (DCE) from the Open Software Foundation (OSF)

c. Common Object Request Broker Architecture (CORBA) from the OMG

d. Basic Reference Model of Open Distributed Processing (ODP) (ISO/IEC 10746)

The ANSA effort is the oldest of these, and the ANSA participants have played major roles in the definitions of all of the others. As a result, the basic concepts are quite similar, but the different membership structures of the sponsoring groups have led to a different focus in each group. The ANSA effort includes an implementation that has been used largely as a proof-of-concept testbed, but has also been used in at least one production system by NASA. The other three are discussed in more detail in the paragraphs below.

It should be noted that OOT, while neither necessary nor sufficient for distributed computing, generally facilitates distributed computing because of the way it encourages encapsulation. The object paradigm generally encourages a client-server view of the relationship between one object and another. While client-server computing is, again, not synonymous with distributed computing, it is one way to provide the natural separations that permit transparency mechanisms to be implemented.

3.4.3.1 Distributed Computing Environment (DCE)

DCE is a distributed client-server environment developed by the OSF. The process by which DCE was developed differs from other efforts in that it was based entirely on existing technology. OSF invited vendors of COTS products to supply services in several areas. The vendors had to be willing to supply source code and to demonstrate the working of the product. The proposals were then evaluated and one product was selected for each area. OSF then integrated these products into a working system.

State of the Technology

The OSF is presently marketing a specification and a reference implementation of DCE and is working on a specification and a reference implementation of an extension called Distributed Management Environment (DME). The OSF also announced its intention to adopt the CORBA standard in a future version of DCE and DME. X/Open is working on the adoption of the DCE specification as part of its set of open specifications as well as CORBA.

Assessment

The DCE is not intended to be object oriented. The DCE is considered more client-server oriented, but the distance between client-server and object-oriented with respect to distributed computing is a subject of considerable debate. For example, there are those that argue that inheritance makes no sense in a distributed system, and to them, the distinction between client-server and object-oriented is largely a choice of words. Others disagree, and to them the adoption of DCE would mean that OOT was not adequately supported, thereby leaving a policy issue if one wanted to do "real" distributed object-oriented computing.

3.4.3.2 CORBA

An object request broker (ORB) is a program that provides a location and implementation independent mechanism for passing a message from one object to another. Objects are registered with the ORB, providing details on the services the object accepts and the invocation mechanism for invoking those services. The ORB then accepts messages for registered services for registered objects, locates the objects on the same or a different platform, and passes the message to the object. When multiple platforms are involved, each is said to have its own ORB that interoperates with the others.

It should be noted that the ORB is not envisioned to serve as a clearinghouse for all object interactions, but primarily for large scale object components, particularly across a network [TAY92]. In fact, the basic function of an ORB does not necessarily require the invoked object to actually be an object in the object-oriented sense. One of the advertised benefits of the ORB concept is that it can be used as a way of encapsulating legacy systems to permit them to be invoked by object-oriented objects. This concept can be generalized to the point where the major benefit of ORB technology in the near term may very well have nothing whatever to do with OOT, but rather serve as the missing link in the arbitrary interconnection of disparate systems.

State of the Technology

The OMG has recently adopted the CORBA as a specification of the architecture for ORBs, and several vendors have already announced products that conform to it. As noted earlier, OSF has announced its intention to adopt CORBA in a future release of DCE. Version 1.1 of the specification is somewhat limited in that it does not require interoperability between the ORBs of different vendors. This problem is intended to be corrected in Version 2.0 of the CORBA specification. Until this happens, all cooperating ORBs in a network will have to be the same brand. X/Open is also adopting the CORBA specification as part of its set of open specifications.

Assessment

As one might expect, experience with building and using an ORB is limited. Still, the systems that have been built appear to offer great promise for the technology. The major concern is the impact of the ORB on the overall performance of the system. As a result, performance has been one of the areas implementors have been concentrating on, and it appears that substantial progress has been made. Whether or not the performance will be adequate for production systems remains to be seen.

3.4.3.3 Open Distributed Processing (ODP)

ODP is intended to become a reference model for distributed processing in the same mold as the Open Systems Interconnection (OSI) reference model for communications. As a reference model, ODP defines classes of services and provides a structure for a hypothetical set of components that will be needed to provide those services. That is, the intention is specifically to be a framework within which to develop standards for services necessary to provide a wide range of transparencies in computing, and is not itself a standard for any services.

State of the Technology

ODP is not yet an approved International Standard, and is not expected to become one for a few more years. The current draft is still incomplete, but its basic structure has been defined. OMG has established a liaison with ISO to converge ODP and CORBA.

Assessment

ODP is largely object oriented to begin with. If it is adopted as the DoD reference model for distributed computing, the inclusion of OOT in the distributed computing services area of the TRM would be automatic. It will be several years, however, before implementations are expected to be available.

3.4.3.4 Summary Assessment of Distributed Computing

The integration of OOT and distributed computing technology is so immature that the very nature of the relationship is still a matter for debate. Experience with distributed computing in general and object-oriented distributed computing in particular is very limited. A number of university and industrial research projects have developed most of the needed concepts and mechanisms to the point where it is believed that production quality systems can be built. A few organizations that see their requirements as compelling have actually built or are building the first of these systems, but the numbers are very small. For example, DCE and CORBA have been on the market for only a short time.

Implementations of DCE are currently available on several computers, but the number of language bindings available for each implementation is small, generally only one. Still, vendor interest seems to be very high. In particular, Unix International, the consortium that defines the requirements and specifications for Unix implementations, has recently decided to add a CORBA-compliant ORB to its 1993 road map.

The three candidate technologies described here are not entirely unrelated. ODP is the most complete reference model. DCE and CORBA both represent technologies that start the process of populating the reference model with component standards even before the reference model itself is complete. As noted earlier, DCE and DME will make use of CORBA in the future.

ODP has not been approved as an International Standard. Additional components are likely to be added. The standardization process currently in effect for DCE and CORBA does not exactly follow the approach recommended for open standards. If the current proponents of DCE and CORBA offer their efforts to the international standards community, they will have to relinquish control. Since these are primarily vendors, it is not yet certain that they will do that. On the other hand, both DCE and CORBA have been adopted by X/Open, which has a level of legitimacy that is closer to that of the international standards community.

3.5 DISA/CIM REUSE PROGRAM

The DoD currently has a Software Reuse Initiative (SRI), which is being carried out by the DISA/JIEO/CIM Software Reuse Program Office (SRPO). The SRI is promoting a strategy of systematic reuse, "where opportunities are predefined and a process for capitalizing on those opportunities is specified" [DOD92]. The SRI is considering the types of domains appropriate for reuse, the types of products, the acquisition and development process, the associated business models, metrics, and standards to support software reuse. A key element of realizing software reuse will be the definition of software architectures, which result from domain-specific analyses.

Within DISA, the SRPO is also implementing a reuse program to serve the IM community within DoD. The SRPO considers OOT as part of its general strategy for accomplishing reuse and is using an object-oriented approach for domain analysis. In addition to OOT, the SRPO has proposed four basic principles to promote software architectures and domain analyses for its community:

- Support the analysis of problems and systems in specific domains. Exploit these domains to support reuse-in-the-large by building domain specific tools such as program (application) generators, very high-level languages, and domain-specific reusable component parts (source code libraries).
- Ensure that reuse is treated as an integral part of software engineering.
- Employ reuse-oriented flexible architectures, with the goal is to achieve "black box reuse."
- Provide the ability to locate and share reusable components across domains and among services. Provide a network of interconnected reuse library systems.

DISA/CIM Reuse Repository

The SRPO sponsors and manages the Defense Software Repository System (DSRS) for storage of its reusable software components. This software library is replicated at four sites around the country, and is available to all DoD agencies and authorized contractors. The DSRS became operational in August 1992, with the transfer of the Army's Reusable Ada Products for Information Systems Development (RAPID) program to DISA. This transfer included 1,600 Ada source code packages, with the largest package containing 300,000 lines of code. The library now contains 2,580 modules of Ada, Pascal, Fortran, and C++ source code, and textual descriptions of source code stored elsewhere. In total, the library now holds approximately 2.2 million lines of code.

To access the library, users enter the system via Internet or modem and enter their needs from a menu of ten categories. There is no browser through an Object hierarchy as is becoming standard in object-oriented languages. Rather the user picks criteria from the 10 categories, and the system searches for a match in the repository. Table 1 contains the 10 categories which describe the language and the functionality of each component.

Table 1. Repository Facets

Facet	Description
Component type	Module type, such as design or implementation
Function	Process that the software performs, such as sort, assign
Object	The conceptual object represented, such as stack, window, person
Language	The language used, such as Ada, Cobol, Fortran
Algorithm	Any special method used with the software, such as bubble, hash, etc.

Table 1. Repository Facets

Facet	Description
Data representation	Data structures, such as record, pointer, arrays
Unit type	The language specific unit, such as file, procedure, package
Certification level	Ninety percent of the components presently in the repository are at level 1 (lowest level - no criteria). Level 4 means compiled and tested by DISA.
Environment	Hardware needed to run the component
Originator	A reference to the organization that contributed the software

Components in the DSRS are classified with a faceted classification scheme. A faceted system does not rely on a hierarchical breakdown of a universe, but instead builds up and synthesizes from the subject statements of particular documents. A faceted system is fluid, and new top-level divisions (facets) can be added as the need arises. In the present DoD repository, there are 10 top-level facets, and 1,300 second level divisions (terms) distributed under these facets.

Assessment

The existing repository can store and retrieve object-oriented code and analysis, design, and prototype models, but the current faceted scheme offers a limited search mechanism when compared to object/class libraries and browsers that are commercially available. In the DSRS, there is a browser that allows a user to see those components that have a defined relationship with a specific component. With the Ada components, it is common to see dependency or "withing" relationships identified between related components. In a similar manner, components with a parent or child relationship can be identified. It should be noted that this capability is provided by the DSRS software and not the faceted classification scheme. For relationships between components to be identified, they must be specified by the repository librarian when the component is entered into the library. In other words, these relationships are not established automatically. However, the DSRS does allow transitive dependencies so that "grandparent" or "grandchild" classes can be identified, but these must be explicitly defined by the librarian. The search capability is limited in that the DSRS does not allow a search based upon a specific relationship a component may have with other components.

To enhance component reuse, the DoD may also want to consider developing standard library classes for Ada9X and graphical class browser support as part of the DSRS capability. Commercial object-oriented languages such as Smalltalk and C++ contain a set of library classes that are shipped as an intrinsic addition to the language standard. Such libraries include the input/output class library, the math libraries, and a library for the graphical user interface.

3.6 DOD DATA ADMINISTRATION PROGRAM

The mission of the DoD Data Administration program is "to provide for effective, economic acquisition and use of accurate, timely, and shareable data to enhance mission performance and system interoperability" [DOD92a]. Its scope extends to all levels, including "installation/base, strategic, tactical, theater, research and development and administrative support programs." The Data Administration program is seeking cost reduction and interoperability through the standardization of data element names and definitions. Its goals are to have a centrally controlled, DoD-wide data repository, standard data, and the use of common procedures and tools. It is currently using IDEF1X for data modeling and wants data elements to "convey a single, information concept" as required by DoD 8320.1-M-1, DoD Data Element Standardization Procedures [DOD93].

The definition of data elements is achieved through a "model-driven" or entity-driven approach. Entities are derived first, then their associated attributes, as opposed to defining attributes and then grouping them based upon functional dependencies. A data element is a "named identifier of each of the entities and their attributes that are represented in a database" [DOD93]. The data element consists of a "prime word" (and modifiers) and a "class word" (and modifiers), with optional property modifiers. The class word with the modifier designates the category and data type to which the data element belongs. Currently, there are 17 predefined class words defined in DoD 8320.1-M-1.

The definition of each data element includes the element name, the type of data (alphanumeric, alphabetic, numeric, integer, etc.), as well as a listing of all possible domain values. The names and domains are intended to be according to logical, not physical, considerations. Data elements do not include any connotations regarding technology (hardware or software), physical location (databases, files, or tables) or function (systems, applications or programs).

Data Administration for OOT

Object-oriented technology collects several data elements into one unit, called a class. Instances of each class are called objects, and each object contains specific data element values. As an example, consider a typically complex object like an employee, with a name, address and salary. In the example below, the name and address are independent classes used by the employee class to store data.

```
CLASS EmployeeClass
-- attributes
            EmpAddress: AddressClass
             EmpName: NameClass
               Salary: integer
-- methods
                SetSalary()
                GetSalary()

CLASS AddressClass
-- attributes
          Street, City, State: CStringClass
              ZipCode: integer
-- methods
               SetAddress()
               GetAddress()
```

This simple example shows the three kinds of attributes present in object-oriented languages:

a. The primitives in the programming language (integer, float, character, reference, pointer, etc.) as noted by the Salary attribute in the EmployeeClass.

b. Predefined library classes that can accompany the language. The CStringClass in the AddressClass above is meant to demonstrate such an example. This class could have been written by the programmer, but a String class is usually available in a predefined library.

c. References, pointers or incorporation of other complex objects. The EmployeeClass above includes two other user-defined classes as members, AddressClass and NameClass.

This example also lists the two most basic methods, one to Set() the value of a simple attribute, and one to Get() the value of the attribute. Every simple attribute such as Salary needs at least two basic functions, Get() and Set() to manipulate the attribute.

In all commercial object-oriented databases now on the market, persistent storage is at the attribute level. In these products, an object with nested child objects is stored as an ordered sequence of parent attributes followed by an ordered sequence of child attributes. The order of storage and the method of storage is vendor dependent, and no standard order of storage has emerged.

A complex object can be decomposed into simple attributes for persistent storage. These attributes can be managed with the current Data Administration program, and there are no substantial changes in Data Administration needed for object data management.

Built-in class libraries, like the CStringClass above, can also be managed by current Data Administration procedures. These complex objects again decompose into simple attributes which can be controlled by Data Administration at the data element (attribute) level.

Assessment

For object technology, data management can still be at the primitive data element level inside each class: every complex object can be unraveled into the primitive data elements of its components, and management of these primitive data elements (object attributes) can be accomplished with the current Data Administration procedures in the DoD.

In addition, building class libraries and managing object data are still relatively new technologies. Early users of object technology have needed several pilot projects to properly define class structures and class methods. At this stage in the development of object technology, operations and methods on primitive data elements are too variable by domain to rigidly define. Since classes and appropriate methods cannot be easily predefined top down, it is best to provide data administration at the primitive data element level.

One issue that has arisen is how data and reuse repositories might be integrated. As noted previously, the current Data Administration program coordinates the development of standard format and naming conventions for exchanging data across organizations and domains. This program does not take an object-oriented approach (i.e., managing object/class definitions); instead it uses the IDEF1X data modeling to identify data entities and attributes that are common across different DoD business areas. The IDEF1X entities and attributes provide the basis for data element names and definitions. The current data elements could serve as a source for object/class attributes. The data elements, however, remain distinct from object/class models and the abstractions defined for the domain of interest.

If the Data Administration program were to take a pure object-oriented approach, then instead of managing data models and data elements (attributes), it would manage object/classes with their associated attributes and operations. This repository of object/classes would look very similar to a reuse repository of object/class definitions. Under this scenario, the function of data administration and reuse would be synonymous. The object models in this case would provide the source of standard data elements. This standardization of data would not have to be implemented internal to the object, but rather could be an external format that cooperating systems could both use and that would be managed by a central data administration authority. Standard data elements could be viewed as domain-independent objects that have concrete external representations and only a minimal set of generic operations (methods). Separate domains would be expected to define their own internal views and representations of these standard objects and provide the necessary conversions to and from the external exchange format. Most likely, they would also enrich and extend the set of domain-specific operations.

3.7. DISA/CIM METRICS PROGRAM

3.7.1 Background

Metrics are essential to understanding, managing, and improving the software process. While no metrics categorization has been standardized, a common perspective is to consider the following (not necessarily mutually exclusive) groups.

- **Basic project management metrics**. These include size, schedule, cost or effort.
- **Product metrics**. Most product metrics relate to quality objectives and include measures of complexity and quality of each product in the life cycle - requirements, design, implementation unit (module, class, etc.). Examples of complexity measures which have been applied to traditional design are summarized in table 2. A direct measure of quality is the number of errors by life cycle product. Indirect measures of quality include reliability, usability, maintainability, reusability and other such "ilities".

Table 2. Traditional Complexity Measures

Requirements	function point complexity factors; Cocomo cost drivers
Design	number of design modules, Cyclomatic complexity
Code	number of code modules, Cyclomatic complexity, module size

- **Process metrics**. These are usually developed from product metrics and project metrics and can be used to predict and manage the development process. Examples include amount of "producer" and "consumer" reuse, completion rate of modules, error density (errors per unit size - class, or module or LOC), effort and errors per life cycle phase, amount of effort to make a class reusable, and productivity. These metrics can be integrated into models that predict development effort and time. [KOR93, PFL89].

The use of some of these metrics is largely unaffected by the choice of development paradigm. These include schedule, effort, cost, defects and some indirect quality measures. Development paradigm does impact the choice of other metrics, such as size and complexity. For example, projects using a traditional data driven or process driven software development approach have typically used either function points or lines of code to estimate and measure size. As discussed in section 3.7.2, other size measures may be more appropriate choices for an OO development.

Current CIM Metrics Program

ODASD(IM) has tasked CIM to support the IM metrics effort. CIM's responsibilities include recommending a final set of software metrics to be collected by DoD Information Systems developers. Initially, CIM will support the set of four core measures defined by the Software Engineering Institute (SEI), which include basic project management metrics as well as software problems and defects. The core measures are defined in table 3.

The SEI has also defined a set of checklists to be used by organizations to establish common definitions by defining exactly what is included and excluded in the core measures.

Table 3. SEI Core Measures

Unit of Measure	Characteristics Addressed
Lines of code	Size, progress, reuse
Staff-hours expended	Effort, cost, resource allocations
Calendar dates	Schedule
Software problems and defects	Quality, delivery readiness

3.7.2 State of OOT Metrics

3.7.2.1 Project Estimation

There has been very little published about estimating schedule and effort for OO development. However, from the available literature and discussions with practitioners, we have identified two strategies for project estimation:

Model Based Approaches

Modified Cocomo

Pittman [PIT93] suggests the use of Barry Boehm's nonlinear Cocomo model to estimate resources, and schedule. One difficulty with Cocomo is that it requires an estimate of the system's lines of code (LOC). Pittman suggests two alternatives to obtain a LOC estimate. One approach is to estimate function points and generate an LOC estimate through conversion factors based on empirical studies. [DIS91]. However, some members of the OOT community believe that function points may not work well for object oriented software development. [TAY93], [FIR92], [SEM93]. The second approach is to break the system down into small enough chunks, in this case classes and behavior, to do an accurate LOC estimate. LOC are estimated for each class to carry out its responsibilities. Pittman suggests using the second approach and also adjusts Cocomo's typical life cycle phase breakdown to account for OO style incremental development.

Other than [PFL90], we do not know of any data that has been published to examine the accuracy of using Cocomo for OOT estimation. Pfleeger's study reports that Cocomo does not provide a very accurate prediction of effort for projects using object oriented approaches.

Linear Model

Taylor [TAY93] suggests using simple linear regression to generate resource prediction equations of the form $E = c_0 + \sum c_i x_i$ where each x_i represents one of the following attributes: number of methods, number of component objects, total number of messages (across all methods), number of objects receiving messages (collaborators), total number of parameters (across all messages), and ratio of public to private methods. Past data can help in estimating these coefficients and a prediction model can be built. Of course, this requires organizations to collect enough data to make valid predictions. Taylor is now trying to develop a reasonably sized sample of cross company observations and to try to obtain some statistical results. Again, no case studies are reported in the literature to validate this approach.

Pfleeger Model

Pfleeger and Palmer [PFL90] note that several researchers have found that size estimates are dependent on the technology used to generate the software system. They investigated a software effort estimation model that uses average productivity to estimate actual productivity. Further, the measure of productivity reflects the use of OOT by using a count of objects and methods per person-month. They note that object and method counts can be made early in the development life cycle and refined as the project progresses.

Their study confirmed that estimation models are technology dependent. It found that for OO projects, general estimation models developed for non OO development do not usually perform as well as those designed for OO development. Only Ada-COCOMO and the Pfleeger model have been developed specifically for OO development. Pfleeger and Palmer's findings suggest that, in the absence of more information, the Pfleeger model may be more appropriate than Ada-COCOMO for OO resource estimation.

Heuristic / Rule of Thumb Approaches

Booch feels that versions of Cocomo are not yet usable to predict OOT development. Booch [BOO93] has suggested some heuristics for OOT project estimation based on his experiences:

- The average developer can process (design, create, evolve) about 100 applications specific classes per year.
- For small projects of less than 50 classes, estimate a team of 3 to 5 people for less than 1 year.
- For medium projects of 75 - 400 classes, estimate a team of not more than 12 people for 1 -2 years.
- For large projects in the range of 400 - 2000 to 3000 classes, estimate a team of 100 people for 2 - 3 years.

3.7.2.2 OOT Metrics for Project Management - Monitoring and Control

Basic project metrics

The choice of OOT for a development paradigm should not impact effort metrics - typically staff hours. Nor should it impact schedule metrics, generally milestones set up by project. However, most OOT practitioners believe that an alternative should be sought to replace the traditional size metrics of LOC and function points.

As discussed in section 3.7.2.1, several authors have suggested alternative size measures, such as the number of classes, methods, and possibly messages. As in Cocomo, one can count number of classes reused vs. newly developed separately to more accurately estimate required resources. At the analysis and design stage, Jacobson [JAC92] also suggests looking at the number of use cases (scenarios). Currently, there is no agreement about appropriate OO size measures.

Product metrics

Design Complexity

OO design complexity measures will differ from traditional ones which include number of design modules and McCabe's cyclomatic complexity. Instead, Kemerer has proposed a whole suite of OOD metrics [KEM91]. These include coupling between classes, number of methods per class, depth and width of the inheritance hierarchy. These can be used to assess the complexity (and therefore ease of implementation and maintenance) as well as the degree of "object orientation" of the design. There is growing interest and usage of his suite. However, very little has been published about usage experiences. Sharble and Cohen [SHA93] have used Kemerer's suite to evaluate complexity and "object orientation" of two alternative designs for the same problem, produced with a data driven and responsibility driven approach.

Class Complexity

To measure class "complexity" (analogous to module complexity) - Lorenz, Booch, and Love [LOR93], [BOO93] have all proposed heuristics for average method size and average number of methods per class based on personal experience. For example, Lorenz suggests that for Smalltalk the average method size should be less than 9 LOC.

Quality

Direct measures such as problems and defects should be unaffected by choice of development paradigm.

3.7.3 Conclusions

3.7.3.1 Assessment

Existing models used to estimate effort and schedule are based on past data that may be inappropriate for OOT. Currently, project estimation techniques for OO projects are quite immature. Traditional metrics (e.g. LOC, number of units successfully tested, function points, etc.) may need to either be replaced or reinterpreted (e.g. classes rather than modules as units). Metrics developed with traditional methods in mind do not address concepts like classes, inheritance, encapsulation, and message passing, [KEM91], [FIR92].

Unfortunately, little data has been collected or published on OO metrics. Much more OOT metrics work remains to be undertaken and published, especially with respect to project estimation and design complexity/quality metrics. Researchers and practitioners have acknowledged the problem and are beginning to start research efforts on OOT metrics.

3.7.3.2 Impact on DISA/CIM program

The CIM program is currently based on the SEI core measures. The most significant impact is on the size metric. An alternative to LOC should be suggested as soon as some consensus is reached in the OOT community. The list of labor classes suggested to measure effort may also need to be revised to reflect activities more commonly associated with OO development - e.g. Domain Analysis.

3.7.3.3 Recommendations

The DoD should support ongoing and new research in OOT metrics, especially in the areas of size, project estimation and design complexity. Also, the DoD should launch an initiative to identify a core set of practical OO metrics, comparable to those defined by the SEI for traditional development. Appropriate measures are extremely important for project estimation and management. Further, we need to encourage the capture of baseline metrics data to facilitate calculation of Return on Investment data and to serve as a quantitative basis for assessment of OOT project suitability. Without adequate metrics, we will not be able to really know the impact of object technology on our business, and it will be much more difficult to justify the transition costs to senior management. [TAL93]

4. TECHNOLOGY TRANSITION

The use of OOT does not automatically guarantee more successful software development projects. OOT must be transitioned to an organization with regard to its existing environment, the training and experience of personnel, the type of systems to be built, and the anticipated opportunities for software reuse from future projects.

To understand the experience of transitioning to OOT, representative samples of OOT transitions were examined. These case studies are provided in Appendix A. Most of the case studies include experience from more than one system. We looked for samples that were MIS-type applications, that were moving from Cobol to an object-oriented environment, and, if possible, were implemented in Ada. Three of the five case studies are from private industry: Brooklyn Union Gas, American Management Systems (AMS), and Systems Research and Applications (SRA); the fourth is from the Software Engineering Laboratory (SEL) at NASA/Goddard Flight Center; and the last is the Base Level System Modernization (BLSM) project at the U.S. Air Force Standard Systems Center. From an analysis of these case studies and technology transition literature, some basic strategies have been identified as a means for transitioning to OOT.

4.1 LESSONS LEARNED

The infusion of OOT into development organizations has not been made without some mistakes. There are "lessons learned" that should be regarded as part of any technology transition strategy. This section looks at a sample of lessons learned in the following areas:

- Technology effectiveness
- Reuse
- Training
- Roles and responsibilities
- Tool Support

4.1.1 Technology Effectiveness

The first and probably most significant lesson is that OOT is not a substitute for software engineering principles, and the promised benefits of OOT will happen only if sound management and engineering practices are in place. Page-Jones recommends that an organization should be "circumspect" about adopting OOT. And that if developers are adopting OOT because they have had difficulties using structured techniques, then they will probably have difficulty using object-oriented methods. He also notes that OOT requires a high degree of engineering discipline beyond even that practiced by experienced software engineers. He stresses that without proper consideration and planning, the use of object-orientation will be relegated to the "Method of Great Hope for about a year" [PAG92]. The effect of "object mania" was cogently described by Burton Leathers of Cognos, from its less than successful experience in using OOT [LEA90]:

> Nonetheless, the fact that we were using an OOPL [object-oriented programming language] was important because it contributed to an attitude which would not otherwise have existed. It was very true and is still somewhat true that OOP protagonists are true believers. The very real benefits of using OOP are presented in a very one-sided fashion which too often leads to the view that OOP is a panacea. This better than life outlook induced a euphoria in management which caused suspension of the normal procedures and judgment criteria. As a result, the product was inadequately defined, schedules were unrealistic and the design process was wholly inadequate. As difficulties manifested themselves, panic response set in and all the problems so graphically described by Brooks in The Mythical Man-Month appeared. The final insult was that the lack of adequate management controls caused the project to carry on—at great expense—far beyond the time it should have died.

The second point regarding effectiveness, is that the "right" design and analysis methodology is not a settled issue. AMS notes that "object development is widely misunderstood" and that "no single methodology, by itself, was sufficient for large-scale transaction processing systems." It also noted that all notations had "pros and cons"; however "all will work" [AMS93a].

The third point was that domain engineering (i.e., defining relevant objects in the application domain) cannot be done in the "abstract" [STA93b]. Identifying and designing classes, particularly reusable classes, requires problem domain expertise, not just methodology expertise. An organization must complete a number of projects before the domain is sufficiently understood for building class libraries. Most early users of object technology have found class design to be an iterative process. NASA/Goddard found it necessary to build one complete application before classes could be finalized for each domain. Other early users have had similar experiences [LEI93, BRO92]. Pittman [PIT93] also warns against relying on the "domain idiot," someone who is versed in object concepts but knows little of the domain.

4.1.2 Reuse

The benefits from reuse are often cited as the major reason for adopting an object-oriented approach. These benefits, however, may not be realized until several object-oriented projects have been completed and reuse libraries have been built. Page-Jones [PAG92] notes that the "benefits from reuse will not be immediate" and that an organization should "beware of inflated productivity claims." Reductions in lines of code will be the result of having a "sound in-house library and about five years of elapsed time." AMS estimates that it will take one to two years before there are payoffs in reuse [AMS93].

In addition to managing expectations, reuse must also be planned for. Building reusable software requires organizational commitment and engineering discipline. NASA/Goddard's substantial success with reuse resulted from making reuse a primary concern and not just a by-product of its engineering process [WAL93a]. Because reuse must be considered at design, an organization should expect a substantial overhead in making a product reusable. Page-Jones [PAG92] estimates that "it takes about 20 person-days per class to build for the here-and-now," with "about 40 person-days per class to build in solid reusability for future projects."

SRA found that reuse across its Ada systems was limited because of different software architectures and implementations, with different DBMSs and with different bindings and GUIs (graphical user interfaces). It found that reuse was less attributable to inheritance and verbatim reuse than the reuse of code templates [SRA93c]. In a similar manner, the BLSM effort appears to have substantial reuse by developing standard screen templates [BLS93a].

4.1.3 Training

While the potential payoff for using OOT is high, learning the object-oriented paradigm may be a very challenging task, particularly for those developers with years of functional-based development experience. Learning how to identify objects requires time and training, and object-oriented methodologies vary in their degree of guidance. Both the technology transition literature and the case studies noted that there will be a significant learning curve, with developers needing at least six months to one year to become proficient [PAR92]. Page-Jones [PAG92] estimated that "a shop of over 100 people will need at least 3 years to convert to object-orientation." NASA estimated that an organization should plan for a 5 to 10 year conversion [STA93].

The timing and type of training will also have affect the success for OOT conversion. In a number of case studies, developers were trained in Smalltalk as an initial step in learning object concepts. In addition to learning Smalltalk, training in software engineering concepts was also considered essential. For maximum benefit from training, it should be given just prior to actual use or "just-in-time" training [PAR92, BLU91]. Taligent [TAL93] recommends a "total immersion" approach to learning object-oriented concepts. In addition, object concepts need to be used and exercised during training; it is recommended that students produce an object-oriented product from their training.

One additional concern regarding training is that it is necessary for those other than the development team, such as customers and management, to understand object-oriented concepts. Specifications for requirements and design that use object-oriented notations may not be easily understood [WAL93].

4.1.4 Roles and Responsibilities

The use of OOT may require the establishment of some new roles and responsibilities that are specifically suited to OOT, such as domain expert/analyzer, class architect, object/class librarian, and reuse tester/specialist [AMS93a]. The domain expert is someone who is knowledgeable in the application domain, not necessarily however with the object modeling and software engineering aspects.

An organization may also want to establish a "technology receptor group" as a means of transferring technology within the organization. In both the AMS and BLSM cases, a specialized core group was trained and then later used as mentors to the developers who received the next round of training. Parkhill calls this a "Technology Receptor Group"; it essentially serves as an entry point for new technologies for an organization. The BLSM effort created a "Cadre" of government personnel who served as the mentors. The initial AMS team provided a focus for OOT expertise. The important quality to note here with the mentoring groups is that technology transition is recognized as a human activity. A major obstacle is getting the developer who has been implementing code in a procedural mindset to understand the object-oriented paradigm. This transition takes practice, time, and a high degree of interaction with mentors.

4.1.5 Tool Support

As with the use of almost any methodology, the success of OOT will be enhanced if there are software engineering tools to support the chosen object-oriented methodology. This is especially true for object-oriented methodologies since these approaches make extensive use of graphical notations. Managing and updating these notations often require automated support. In the JIAWG (Joint Integrated Avionics Working Group) experience in using OOA, the lack of a CASE tool to support the Coad-Yourdon OOA approach was a serious impediment to its success [BLU91]. Fayad et al. [FAY91] also note that it is necessary to get CASE tools that support a project's particular object-oriented method.

Another observation regarding many of the object-oriented analysis and design CASE tools was that developers of methods are also vendors of their own proprietary tools. This situation is different from the structured methodologies market, where a set of common structured methodologies is supported by several tools [MAZ92]. As noted earlier, a couple of the more popular object-oriented methodologies such as Schlaer-Mellor and Rumbaugh are now supported by more than one tool vendor.

4.2 GENERAL TECHNOLOGY TRANSITION STRATEGIES

Based upon the experience papers and case studies, there appear to be three basic strategies to transitioning to object-oriented technology: (1) system/project based, (2) domain based, and (3) enterprise based. In the first strategy, OOT is adopted as the result of the needs of a specific system development, not as a corporate or domain-wide adoption. The system-based strategy was more typical of Brooklyn Gas and SRA. In the domain-based strategy, OOT is used across multiple systems and projects but within the same domain as typified by NASA/Goddard's experience. In the third case, OOT is adopted enterprise-wide as part of a corporate strategy for system development. In this broader strategy, there are considerations for standard architectures, methodologies, and processes. This type of situation was more evidenced by the AMS and BLSM efforts.

4.2.1 System/Project-Based Strategy

The system/project-based strategy is driven by the demands of a specific project. This may be the result of imposing a design or implementation requirement (such as language) that is better suited to an overall object-oriented approach. Or its use may be the result of the specific motivation of the development team that has an interest in using OOT. This use of OOT is more "bottom-up." There is no requirement for all domain or enterprise-wide software development to be object oriented. The degree of reuse may vary according to the specific reuse goals of the organization.

4.2.2 Domain-Based Strategy

A domain-based strategy is one in which the use of OOT is not mandated enterprise-wide, but has been adopted for a specific domain of applications. In this approach, there is planning for reuse since multiple systems will be developed within the domain. The NASA/Goddard SEL falls into this category, since OOT is not required but has been incorporated into systems development which is relatively confined to the flight dynamics domain. Even before NASA started using OOT, it was planning and building for reuse with components, specifications, and architectures.

4.2.3 Enterprise-Based Strategy

In the enterprise-based strategy, the implementation of OOT is adopted throughout the software development enterprise. In this sense, OOT is more mandated rather than just available. The use of OOT with other changes may be seen as a competitive edge, such as the ability to get products to market quicker or to respond to customers needs faster. The use of OOT is expected for the long term and there exists the expectation of the future use of "objects." Hence, there will be an effort to build reusable libraries and architectures. This type of transition is best illustrated by AMS. Although it wasn't truly enterprise-wide, the BLSM case is grouped here because it took a broader perspective than just the system or domain, building reusable classes across three different domains. Both AMS and BLSM adopted OOT as a larger strategy, not just based upon the requirements of a specific system development.

These transitions contained elements of the following transition strategy recommended by David Parkhill [PAR92]:

1) Start small: A company should choose a project that is low risk, but real, not a toy project. One danger with embarking on an OOT transition is that OOT is oversold to management and expectations are set too high. The result is that the early use of OOT is on high visibility, high risk projects. Since the use of OOT introduces a major factor of complexity, other risks should be minimized. In a similar manner, an organization should use only those portions of technology that are sufficiently mature [PAG92].

2) Train a core group: Training a core group allows the transfer of technology to happen "through people and mentors" [PAG92]. Parkhill calls this a "Technology Receptor Group," and it should be set up six months to one year prior to production use. This core group should consist of developers who are quick learners and highly motivated to learn OOT. They will serve as teachers/mentors to the organization's developers. Everyone should receive object training, but a core group may receive additional training. Members of this group often serve as mentors to small development teams.

3) Use the core group as mentors: The establishment of a mentor group reflects the need for solid, in-house working OOT expertise and its availability to new developers learning the object paradigm. Noted over and over with this technology is that it requires a different perspective in problem solving; and that shift in thinking may not happen as the result of a one-week course, but takes practice and with guidance from OOT experts.

4) Plan for a substantial conversion time: An organization should anticipate and plan for a substantial conversion time for a OOT transition. The core mentor group should be set up six months to one year prior to actual production use [PAR92]. Page-Jones [PAG92] notes a 3-year conversion process for a shop of 100 people.

5) Identify testbed applications and start prototyping: One essential aspect of this technology is that it must be used and practiced. In both the AMS and BLSM cases, organizations conducted prototyping (in Smalltalk) before actual system development. This opportunity to do hands-on work helps developers to refine their understanding of object-oriented concepts.

6) Measure key process indicators: It is important for an organization to know what works and what doesn't. It should measure such factors as design and implementation time, component reuse, and error rates.

4.3 TECHNOLOGY TRANSITION FRAMEWORK

Object-oriented technology should be introduced according to the maturity of the technology and the environment. Certain aspects of OOT are more mature than others. For example, object-oriented programming languages are more mature than object-oriented database technology. In addition to the maturity of the technology, the environment into which it will be introduced will be a significant factor to its operational success. The organization may have policies and programs that effectively prevent the use of a new technology. Or there may be a lack of training and tools for software engineers. Whatever the technology, there are a number of factors that will contribute to a technology's success or failure.

Table 4 is a proposed framework for illustrating suggested stages of OOT technology infusion into an organization. This framework has five stages:

- Impeded
- Passive
- Active
- Optimized
- Coordinated

Impeded: The initial state of the transition model is considered "impeded" because existing policies, standards, and dependence on other technologies impede the widespread use of the new technology. Use of the technology happens under special conditions, where exceptions or waivers are granted to standard policies and procedures. The use of the technology conflicts with instituted methods for acquisition and development.

Passive: In this stage, the technology does not have any major impediments or overriding technical risks. Its use, however, is not encouraged either through policy or tool and training provisions. Its use may be in pilot projects, which at this stage are not required to receive exemptions to existing policies and standards. Tools and training are acquired in an ad hoc, project-specific manner.

Active: Tools and training are widely available and supported, and the use of the technology may be encouraged. Use may still be system or project specific. There exists an environment where users can gain experience with the technology and a variety of methods and approaches may be employed. No institutionalization of the technology yet exists.

Optimized: In this state, use has accelerated and improved to the point where as a community, there develops some consensus regarding the value of different approaches and methods. By this point the use of the technology is not new and there is a well-developed community of experienced practitioners. Technical issues have been resolved to secondary or tertiary status. There is the development of reusable class libraries. The technology can start to be exploited in innovative ways and there exists the opportunity to pursue common definitions and interproject coordination.

Coordinated: There exists sufficient experience to agree on standards, protocols, and common methods. The variety of approaches in the technology may have evolved to a closer consensus by this point. This should be evidenced by common definitions. Use is enterprise-wide, with common established policies, procedures, standards, and building blocks. Opportunity exists to integrate with other technologies.

This framework is intended to be applied to specific enterprise and organizational actions and is intended to accommodate the varying maturity of OOT and the varying abilities of different organizations to adopt it. For the DoD, the enterprise-level actions are those to be undertaken at the OSD level and would be effective DoD-wide. The organization-level actions are those specific to an individual organization, such as a Central Design Activity, considering or undergoing a transition to OOT. Ultimately, the actions taken should help the organization or enterprise move to the next stage in the transition framework.

Table 4. Technology Transition Framework for OOT

Stage	Characteristics	Transition to Next Stage	Enterprise Actions	Organizational Actions
Coordinated	- Standards - Integrated - Interconnectivity - Interproject coordination - Architectures	Integration with other technologies	- Monitor OOT and other technologies	- Keep skills and tools up-to-date
Optimized	- Skilled level of practice - Reuse of classes - Selective use of methods - Process and product measured	Integrate OOT use into (across projects and the DoD Enterprise domains)	- Coordinate a selection of OOT methods - Resolve inter-organization/project impediments	- Develop standard class libraries
Active	- Training and tools in place. - People getting experience - Using mix of approaches - Non-integrated - Project specific - No multi-system/ domain coordination	Assess and optimize OOT use	- Assess OOT use across enterprise - Fund early domain-based use - Disseminate OOT assessments and information - Identify meaningful OO based metrics	- Evaluate approaches to OOT - Look for opportunities for domain-based use of OOT
Passive	- No policy impediments - No major technical risks - Ad hoc - Non-integrated - Project specific - Pilot projects - No organized training and tools	Introduce OOT	- Fund early projects - Establish training policy	- Acquire technology - Conduct pilot projects - Get tools and training
Impeded	- Impediments still exist - Maybe some exploratory or experimental work - nonoperational - Specialized projects on waiver	- Eliminate or minimize technical, economics, policy, cultural impediments	- Technology assessments - Enterprise needs assessment - Analysis of policy - Revise policy, standards, programs	- Conduct exploratory projects - Identify technical risks and issues

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

In considering whether the DoD should implement OOT within its information system development, we have reached the following conclusions:

OOT appears to offer advantages over traditional process-driven and data-driven approaches. The features of encapsulation and inheritance support increased maintainability and reusability which leads to lower costs in software development and maintenance. OOT also appears to facilitate distributed computing through the use of encapsulation which encourages a client-server perspective.

OOT is not a completely mature technology. There still exists a diversity of concepts and notations used in development methodologies and a variety of mechanisms for object database technology. For now, relational database technology should remain the primary method for simple business data, with object database technology more appropriate for complex data types.

There appear to be relatively few, if any, impediments within DoD AIS lifecycle management and software development standards to using OOT. DODD 8120.1/ DODI 8120.2 and the anticipated DOD-STD-SDD do not present any impediments to using an object-oriented approach. Both support iterative and incremental lifecycle development models. The existing DoD lifecycle process model for AISs and the anticipated software development standard should accommodate the use of object-oriented technology.

There are impediments and technical risks within the DoD Process Model for IM and Technical Reference Model that should be resolved prior to any large-scale transition to OOT. The Process Model for IM may need to revised to resolve the IDEF0 and IDEF1X to object-oriented transition between functional process improvement and system development. The TRM may need to be revised to accommodate object-oriented database technology.

A successful transition to OOT will require tools and a significant training effort. Since the success of OOT will depend in large measure upon the capability of the individuals who use it, training in object concepts along with software engineering concepts is essential, as is the support of automated tools.

5.2 RECOMMENDATIONS

The overall recommendation of this effort is that the DoD should support the introduction of OOT as part of its software engineering and information technology strategy. The introduction of this technology will have to consider the large-scale consolidation of DoD information systems currently underway, their migration to open standards, and ongoing reengineering efforts. The integration of this technology will also need to consider functional process improvement activities, software development process and standards, legacy systems, and emerging technical and functional architectures. The move to OOT should be based upon a number of factors: the maturity of the technology, the ability of the environment to accept and use it, and the economic justification that such a transition is warranted.

The following are some specific recommendations regarding the implementation of OOT in the DoD. These recommendations are grouped according to the four transition stages defined in the Technology Transition Framework in Table 4.

Risk Minimization and Impediment Resolution

Develop a technology transition strategy for the introduction of OOT: A technology transition strategy will establish priorities for transition activities, define the scope and stages for the transition, the roles and responsibilities, and the mechanisms necessary for a transition to occur. This strategy should meet the mission and information technology needs. It should also consider the technology transition approaches described in Section 4.2.

As part of a technology transition strategy, conduct a business case analysis for transitioning to OOT. This transition could directly affect over 40,000 DoD software developers. Nearly all reports regarding technology transition suggest that the adoption of OOT cannot be made without adequate training in software engineering and object concepts. Such an analysis would need to consider the existing base of people and systems and the anticipated opportunities for software reuse. The benefits for reuse appear to be more achievable within a particular domain or functional area. One scenario, for example, would transition projects on the basis of their functional areas.

Develop example scenarios for the use of OOT. It may also be useful to develop example scenarios that would map the use of OOT onto the existing DoD information infrastructure of policy, procedures, programs, and standards. These scenarios would uncover those areas where risk may exist. Although this document has provided an assessment of OOT use for a number of areas, there remains the entire acquisition and development process to consider. A basic scenario would include a "To-Be" model of OOT use that could present in one picture the use of OOT starting from functional process improvement through software development to database development.

Participate in or monitor those organizations developing standards regarding OOT. The DoD has an interest in seeing that standards bodies consider the needs of the DoD. For example, the Object Management Group currently has over 300 member companies and is developing standards for a variety of object concerns, including methodologies, notations, and architectures.

Reevaluate the Process Model for Information Management: Consider the development process, products, roles, and the Process Model for Information Management. The addition of object-oriented analysis and design techniques as mechanisms during application development is relatively trivial. The model should be reevaluated to resolve the shift in paradigms between the Functional Process Improvement activity, which uses an IDEF approach, and the system development activity, which could potentially use an object-oriented development approach.

Conduct controlled studies to resolve specific technical issues. There are a number of specific technical issues within OOT that have yet to be resolved, such as the different mechanisms for distributed computing, the different approaches to object-oriented analysis and design, which combination of organizational, domain, and project characteristics make OOT the preferred development paradigm, and which metrics are appropriate for OO software projects. In addition, specific studies and working experience may help determine a set of criteria for transitioning and applying OOT to particular organizations and projects. The DoD could then encourage object-oriented projects in those domains that fit the criteria for object technology.

Introduction of OOT

Establish testbed sites for investigating and exercising OOT. Investigate the types of problems where object-oriented approaches in analysis and design may be more desirable. Such sites could pursue pilot projects in OOT, with particular emphasis on reengineering legacy systems to OOT.

Make OOT concepts part of Ada and software engineering training. Develop and institute a training program that incorporates OOT as a part of software engineering, Ada, and design and analysis methods. Although object-oriented methodologies such as OOA have a variety of notations, they are still useful in teaching the object paradigm. For OOA and OOD methodologies, the non-object-oriented aspects such as functional and dynamic modeling should also be included. For groups ready to begin an object-oriented project, the DoD may also want to consider concentrated object-oriented training, including other object-oriented languages such as Smalltalk or Eiffel for introducing object concepts.

Do not mandate OOT as an approach for all software development nor adopt any specific object-oriented methodology or technique at this time. Further analysis is needed to determine which object-oriented technique is desirable for application development. The use of OOA alone, for example, will not guarantee a requirements specification that will be correct and complete. Software engineering activities require a variety of techniques, including event and process modeling and prototyping. Likewise with design, OOD may not be the optimal design approach for every implementation. Considerations such as performance and efficiency could conceivably require a non-object-oriented design.

Assessment and Optimization of OOT

Encourage the development of ODBMS bindings for Ada9x. Currently, most ODBMS bindings exist for either the C++ or Smalltalk programming languages. The DoD should encourage the development of ODBMS bindings for Ada9X, including the investigation of emerging standards such as SQL3 and ODMG-93 (Object Data Management Group) specifications.

Encourage domain-based use of OOT. Eventually, the use of OOT should move from beyond the system/project perspective to that of a domain or functional area. It is with a domain-based use of OOT that we expect to see the benefits of extensive software reuse, as evidenced by the technology transition case studies and literature. A domain-based approach may also provide a means of transitioning OOT to an eventual enterprise-wide usage.

Support ongoing and new research in OOT metrics. Since traditional metrics may need to either be replaced or reinterpreted, the DoD should launch an initiative to identify a core set of practical OO metrics, comparable to those defined by the SEI for traditional development. Appropriate measures are extremely important for project estimation and management and without adequate metrics, we will not be able to really know the impact of object technology on our business.

Integration of OOT into the DoD Enterprise

Support efforts towards a commonly accepted definition for OOT. A definition is needed that accommodates Ada and the software engineering concepts that support high reliability and maintainability. This will have an effect upon the mechanisms used for defining objects, particularly comparing the inheritance vs. encapsulation qualities of OOT. Develop a common notation with care to include a robust set of semantics.

Develop a standard set of Ada class libraries. All commercial object-oriented languages contain a set of library classes shipped as an intrinsic addition to the language standard. These classes are usually built-in abstractions such as sets, collections, arrays, bags, and strings. These built-in classes have standard methods associated with them. To aid the development of object-oriented applications in Ada, the DoD should develop a standard set of library classes similar to the predefined classes in Smalltalk and C++. These classes can speed the development of object-oriented systems using Ada 9X.

APPENDIX A. CASE STUDIES OF OBJECT-ORIENTED TECHNOLOGY TRANSITION

A.1 NASA/GODDARD SOFTWARE ENGINEERING LABORATORY

A.1.1 SYSTEM/PROJECT CHARACTERISTICS

A.1.1.1 Background

The Flight Dynamics Division (FDD) of NASA's Goddard Space Flight Center, Greenbelt Maryland, develops two types of software:

1. Ground Support Software. This type of software program receives actual telemetry data back from a satellite and processes it in real-time or batch mode. Real-time processing includes user interaction with via windows and user interfaces.
2. Simulation Software. These programs create large test files that can be used as input to the Ground Support Software in item (1) above. The simulation programs are entered and run in batch mode. These simulations are used to test and develop various algorithms for insertion into future ground support software.

Development of this software occurs as a joint venture between FDD and the Computer Sciences Corporation (CSC), Goddard's primary outside software contractor. For 23 years, the System Sciences Division of CSC has worked with Goddard to develop software for the space program. In 1992, FDD employed about 25 software engineers and CSC employed nearly 100 engineers and programmers. These employment figures do not include administrative and staff support personnel.

Since 1986 FDD estimates it has completed about 30 new ground support and simulation programs (items 1 and 2 above). The average size of program has ranged from 100 KSLOC to 300,000 SLOC. In the mid-1980s, FDD began to experiment with object-oriented technology, and since 1986 11 of these 30 projects have been built using object-oriented development methods. The large number of projects and long time duration probably make FDD the most extensive user of object-oriented software methods in the government.

The Software Engineering Laboratory at Goddard (SEL) has been a catalyst in the introduction of these new object-oriented methods into NASA. SEL is a joint venture of FDD, CSC, and the University of Maryland Computer Science department. Since 1976, this joint venture has been funded and sponsored by NASA. Its purpose is to help Goddard measure and improve the software development process at the Goddard Space Flight Center. Much of the emphasis and study of OOT has come through the SEL.

A.1.2 PURPOSE AND TYPE OF APPLICATIONS

A typical FDD project lasts between 2 and 4 years, and ranges in size from 100,000 to 300,000 SLOC. The software applications can be best described as scientific batch mode software. The software models physical processes, and makes extensive use of matrix transformations, and vector algebra. The simulation programs, item (2) above, run in batch mode with no user interaction. The program results are returned in batch mode as a printout. The Ground Support Software, item (1) above, includes both user interactive and batch mode programs.

The domains of each project are narrow in the sense that they examine a subset of spacecraft control, such as attitude determination and control, rather than the entire spacecraft or ground support system. Each new software project also builds from the software used in the last similar spacecraft project. The 11 completed OOT projects include 3 early Ada ground simulators built between 1985 and 1988, three Fortran ground support systems, and four telemetry simulators built from 1988 to 1992.

A.1.2.1 Size of Systems

During the seven years FDD has been experimenting with OOT, developers have completed eleven projects using OOT methods. These projects have ranged in size from 100,000 LOC to 300,000 LOC, and generally have taken between 2 to 4 years to complete. These 11 projects represent about 30% of all work at the FDD since 1986.

A.1.2.2 Languages

Seventy-five percent of all Goddard software development uses Fortran, fifteen percent uses Ada, with the remainder in various languages. However, for the projects completed in OOT, the Goddard developers primarily used the Ada language. Of the 11 OOT projects, 7 have been in Ada and 4 in Fortran.

Fortran and Ada are not normally considered object-oriented languages, but by using the basic object-oriented principles of encapsulation, abstraction, and modularity, FDD has been able to apply Fortran and Ada successfully in its object-oriented development.

A.1.3 PROJECT MANAGEMENT AND TECHNOLOGY TRANSITION

A.1.3.1 Level of Effort and Duration

A typical object oriented project has taken two to four years to complete. The software development team consists of 5 to 15 people organized into 3 or more subgroups. To develop a project, Goddard tries to match application experts in a specific field, like gyro physics, with software specialists who can translate the physics into structured systems.

A.1.3.2 Staff Skills and Experience

FDD and CSC have been developing space software for the past 23 years. In 1992, there were about 25 government software developers at FDD and 100 CSC developers. These figures do not include staff support or administrative personnel. About 20% of these developers use object-oriented methods.

Introduction of OOT began with the introduction of Ada to the FDD in 1985. Previous Ada experiments had produced designs and code that looked like Ada versions of Fortran systems. So in 1985 the first team was trained in a variety of design methods, including Booch's Object-Oriented Design method, stepwise refinement, and process abstraction. The University of Maryland provided training classes on data abstraction.

FDD's capability to use object-oriented methods has grown in the past seven years. OOT efforts began with the introduction of the Ada language in 1985. Prior to the OOT efforts, data for each simulation was stored in one large global database where it could be accessed by a Fortran program. Initial OOT efforts were to simply break up the single large database into segments, and store the segments with local procedures that acted on each segment. This segmentation of the database proved successful, and led FDD to increase abstraction and encapsulation efforts in other OOT projects. Current efforts are now focused on layered architectures of objects that can act as services to other higher-level objects.

CSC personnel work alongside government personnel in the development of each project. About half of the CSC software developers have been with CSC for over five years and the average tenure for FDD software developers is about six years. Most developers in both organizations are college graduates, but not necessarily computer science graduates. CSC finds that a new hire who understands physics, astronautics, and space vehicles can be trained to develop software for these applications. Understanding the domain is the key element for these scientific applications. Interviews with FDD and CSC personnel indicate it takes two to four years for a development team to reach an expertise level with the OOT technology.

A.1.3.3 General Technology Transition Strategy

The introduction of OOT began in 1985 with the use of the Ada programming language in the laboratory. To introduce the technology, FDD picked those software developers who expressed an interest in object-oriented methods. About 20% of the development staff liked Ada, and wanted to work on object-oriented methods. CSC and FDD picked people from this group.

Object-oriented methods are not mandated for a new project, and Goddard lets each development team decide which techniques best apply to that project.

A.1.4 TECHNICAL APPROACH

A.1.4.1 Lifecycle Process Model

The FDD uses a formal design method based on the "waterfall" approach to software development (requirements, analysis, design, coding, test, and maintenance). It allows the use of either structured design or object-oriented techniques, based on the specific needs of each project.

A.1.4.2 Standards Used

The FDD process for software development has been developed in-house, and is called the "Recommended Approach to Software Development." This set of guidelines allows the use of either structured design or object-oriented methods, based on the specific needs on each project.

A.1.4.3 Methodologies Used

In 1986, training was given in Booch Object-Oriented Design, since it was the only method available at that time. The University of Maryland also conducts classes through the SEL on data abstraction methods. However no formal methodology (data-driven, structured analysis, etc.) is used by FDD. Each project is examined on its merits, and the best methods chosen for that project.

A.1.4.4 Development Environment

Most object-oriented development is done using Ada. Development is done on the VAX/VMS system using DEC compilers and VAX software editors and tools. VMX configuration management software and performance analyzers are used during development. Some Fortran object-oriented development is done on an IBM mainframe with a minimum set of tools, mainly compilers and editors.

A.1.4.5 Reuse of Components

The managers at CSC think that software reuse has progressed through 4 levels over the past 10 years at CSC. In the late 1970s, there was reuse of people, assigning the same development team to the next similar project. Then in the early 1980s, a second-level of reuse occurred as CSC focused on the reuse of code. By the mid-1980s CSC reached a third level, with development of reusable processes for development. Starting in 1989, at the fourth level, CSC is attempting to focus on reuse in the full lifecycle, from requirements to maintenance.

The largest benefit from reuse has been the encapsulation of the database segments with local procedures. This encapsulation has caused reuse of code and segments to increase from about 20% to 70% today. Objects which are common throughout a design or common to several designs are held as standard utility libraries. Other packages call these libraries to obtain needed services.

A.1.5 PROJECT RESULTS

In domains where reuse is possible, OOT methods have enabled a three-fold effort decrease, from 320 staff-hours/KSLOC to about 140 staff-hours/KSLOC. Converting the figures to monthly production, this represents a productivity gain from 50 SLOC/staff month to about 150 SLOC/ staff month today, while project cycle time has been cut approximately in half. During this increase, the error rates have dropped from 4.5 errors/ 1000 SLOC in the early 1980s to about 0.8 errors/SLOC today. Most of the improvement has come from reusing tested code (verified and validated), rather than from improved system testing or improved ease of maintenance.

The FDD introduced the programming language Ada at the same time object-oriented methods were introduced into FDD. Managers at both FDD and CSC feel the major change was the use of the object-oriented methods rather than any particular language. In fact much of the code (75%) is still written in Fortran.

A.1.5.1 Lessons Learned

a. For NASA, the SEL serves as a catalyst for new ideas, and the introduction of new technology. Most of the papers on OOT and the metrics of performance, and some of the training were performed by the Software Engineering Laboratory. This catalyst helped foster the new technology in NASA, gave validity to the new technology, and made the transition easier for the organization.

b. Managers at CSC indicated there is a need to train both managers and software developers in object-oriented methods. Originally, developers were trained in OOT and managers were not. This left the managers at a disadvantage in evaluating the status and quality of development and finished products.

c. The greatest benefits from OOT came from encapsulation and abstraction, not from any particular language or methodology. The introduction of OOT methods in 1986 was more important than the simultaneous introduction of Ada.

d. OOT cannot be mandated from above as organization-wide method. Rather, each project must be evaluated on its own merits for the use of the best design methods.

e. Even in the narrow domains in which it works, FDD found that domain analysis cannot be done in the abstract. There is a need to first complete a project using OOT methods, and then to generalize from the finished project. FDD used a bottom-up approach and developed the project from modules. FDD and CSC managers indicated that the bottom-up approach gave developers a better understanding of the details of the abstractions.

f. NASA used the same measurement and evaluation procedures for OOT as for procedural software projects. There was no need for different procedures for OOT projects. However, there was a need for mapping products and terminology to standard procedures within FDD. For example, "unit" needed to be defined for tracking, testing, and documentation purposes.

g. Not every software developer was interested in the new technology. Only 20% of the developers wanted to work in OOT and in Ada. The interested parties were generally younger, with a need to keep resumes current and a desire to keep abreast of the latest technology. Even after eight years, FDD has not mandated OOT throughout the organization.

h. To develop OOT projects, NASA uses the tools that come integral to the VAX software environment. CSC and FDD found that software tools need to be integral to the platform that will execute the program. In the past several years FDD has tried numerous design and requirement tools. It found that vendor claims of interoperability were often not supported, and the projects could not move easily to execution without fixes and recompilation. Several years ago SEL determined that automated tools compound the design problems unless people first understand the "pen and pencil" methods.

i. NASA has found that the OOT projects run slower than the Fortran-based procedural projects. Studies by FDD and the SEL have found that this problem is not caused by the object-oriented methods, but by a decision to use variable length records for database storage. Originally, OOT developers were told that performance considerations did not matter in this experimental technology. So they focused on cleanness and simplicity of the code. Users of the code, however, complained that speed was slower than the procedural methods. In hindsight, managers now feel that the object-oriented methods should have focused on both the new technology and the speed needed to match existing applications.

A.2 BASE LEVEL SYSTEM MODERNIZATION (BLSM)

A.2.1 SYSTEM/PROJECT CHARACTERISTICS

A.2.1.1 Background

The U.S. Air Force manages a number of its software initiatives at the Standards Systems Center, Maxwell Air Force Base - Gunter Annex, Montgomery, Alabama. One major effort is the Base Level System Modernization (BLSM) program, which is currently migrating a total of 36 standard base-level systems to an open systems environment. Currently, there are three pilot projects being developed under the BLSM program: (1) Logistics Module-Base Level (LOGMOD-B), (2) Air Force Operation Resource Management System (AFORMS), and (3) Manpower Data System (MDS).

A.2.1.2 Purpose and Type of Applications

The LOGMOD-B system is a tool for base level logistics planners for planning and executing mobility support for worldwide deployment of forces. The AFORMS system tracks Air Force flying and ground training programs for each weapon systems at each base. The MDS system manages Air Force manpower requirements and tracks authorizations for those resources [HAR93].

A.2.1.3 System Size

The estimated size of the new LOGMOD-B is 60,000 LOC, with 3,191 function points and an overall 35% increase in functionality. The original LOGMOD-B was 235,000 LOC or 1,885 function points. The original AFORMS was 340,000 LOC of Cobol, with the estimated size for the new system at 6,807 function points.

A.2.1.4 Implementation Language

The implementation language for BLSM projects was Ada83; Ada9X will be used when it becomes available. Smalltalk was used for full function operational prototypes.

A.2.1.5 Hardware

The target hardware platforms, which are Unix based, are the HP 9000-700 for LOGMOD-B and the HP 9000-700 and Unisys 2200 for AFORMS and MDS. The target platform may also move to Intel 80486.

A.2.1.6 DBMS

For the LOGMOD-B, the BLSM effort is using a relational DBMS. The primary reason it is using a RDBMS is that the system is required to be conformant to the TRM which at this time does not specify a data access method for ODBMSs. The BLSM team did investigate the use of an ODBMS (Gemstone) provided by Servio, which worked with the BLSM project to build the prototype for free. With this system, objects, methods, and screens are stored in the database. Methods are coded in a Smalltalk derivative.

The ODBMS did provide a significant improvement to database performance. According to the BLSM team, there was a 300 to 1 improvement using Gemstone vs. an RDBMS. The ODBMS could fetch 30,000 objects per second vs. 100 rows per second. It should be noted that logistics data has complex relationships and the previous implementations of the LOGMOD-B system used a network DBMS.

A.2.1.7 PROJECT MANAGEMENT AND TECHNOLOGY TRANSITION

A.2.1.8 Level of Effort and Duration

Unknown.

A.2.1.9 Staff Skills and Experience

The development team consisted of a mix of government and contractor (Harris) personnel. Their experience and training were originally in non-object-oriented methods and Cobol, with some Ada expertise. For learning OOT the BLSM team initially learned Smalltalk, with the following course of training:

- Object-Oriented Analysis (OOA) Training: 1 week or condensed 3-day class
- Object-Oriented Design (OOD) Training: 1 week or condensed 3-day class
- Polymorphism/Inheritance (PI) Ada Training: 14 days total

A.2.1.10 General Technology Transition Strategy

The BLSM effort made the transition to OOT by setting up a core group that served as a technology transfer/mentoring group. This core group received training in object concepts and then, in turn, served as "object mentors" to new object developers. BLSM also used "just-in-time" training, where individuals are trained just prior to using a technology.

A.2.2 TECHNICAL APPROACH

A.2.2.1 Lifecycle Process Model

BLSM used a "tailored recursive spiral" lifecycle process model. For example, LOGMOD-B is divided into 14 increments, each consisting of classes and objects that are "functionally" grouped. The team built prototypes in Smalltalk to validate data and functional requirements, then implemented the increments in Ada Increments 1 and 2 are currently in the implementation/test phase.

A.2.2.2 Standards Used

The BLSM effort used DOD-STD-2167A, tailoring it accordingly "to accommodate an object-oriented development approach using Ada and an incremental development methodology." Other "standards" used were the DoD 8320.1-M-1 for data element standardization [HAR93].

A.2.2.3 Methodologies Used

The BLSM effort developed its own "object-oriented methodology" which was based upon the Coad/Yourdon Object-Oriented Development Methodology. The object-oriented methodologies of Booch and Rumbaugh were also considered; the BLSM team chose Coad/Yourdon for its simplicity, having one underlying representation. The BLSMA team used Peter Coad for consulting and training. In follow-up interviews with the BLSM team members, they noted that they may use the Rumbaugh approach. Although Coad/ Yourdon was simple, it may not be sufficiently robust in the long term. Also, one application has begun to use the Booch method for OOD.

A.2.2.4 Management of Phase Transitions

OOT was used throughout the development lifecycle except for the implementation of the database which was relational. The BLSM team felt that the transition from phase to phase was much easier with the object-oriented approach. One of the reasons the BLSM team decided to try object-oriented analysis was that it was having difficulty transitioning from a Structured Analysis (functional decomposition) approach during requirements analysis to object-oriented design.

A.2.2.5 Development Environment

The hardware development environment consisted of Unix-based workstations for analysis, documentation screen generation, design, and prototyping; Rational workstations for the design, code, and test of Ada software; and VAX systems hosting a relational database and supporting configuration management. PCs and Macintosh computers were used to support planning, storyboarding, and graphics.

Software tools consisted of Coad's OOA Tool for object modeling and analysis, Smalltalk for prototyping, Screen Machine for screen generation, the Rational Environment and tools for design and coding, and ORACLE for the RDBMS [HAR93]. Other tools were developed in-house such as an Ada Code generator and a tool to export requirements documentation from OOA Tool, reformatted to meet DOD-STD-2167A standards and imported to Interleaf, a desktop publishing software.

A.2.2.6 Reuse of Components

Across the three systems classes were shared through a corporate object model, with a notable example being the generation of screens. The original three systems had 133 unique screens. The BLSM effort created 12 generic screens, from which standard screens with a common look and feel were generated. The resulting standard screen use ranged from 37% to 60% across the systems.

In general, the reuse across the three systems has been limited (15 to 19%). The levels are believed to be low because the three systems are in different domains. BLSM expects to see higher levels of reuse with additional work in the same domains.

A.2.3 PROJECT RESULTS

A.2.3.1 Current Status

The BLSM effort is still early in development, with increments 1 and 2 of LOGMOD-B currently in the implementation/test phase.

A.2.3.2 Benefits

The benefits from using OOT appear to be in the growing levels of software reuse, which are expected to result in reduced development costs. As noted in B.1.3, the size of LOGMOD-B is approximately 25% of the original system in terms of lines of code, but has 35% more functionality. The BLSM team also expects better adaptability and easier maintainability from the new implementation

A.2.3.3 Lessons Learned

a. Configuration control and management are a necessity, particularly when the object models grow in size and complexity. While read access was available to everyone, write privileges to objects had to be limited and controlled [HAR92].

b. One object representation makes the development easier. The BLSM team members encountered difficulties transitioning from one object notation to another. However, they did state that they would like the ability to use any object notation. Ideally, a tool would translate between different notations automatically.

A.3 BROOKLYN UNION GAS CUSTOMER RELATED INFORMATION SYSTEM (CRISII)

A.3.1 SYSTEM PROJECT CHARACTERISTICS

A.3.1.1 Background

Brooklyn Union Gas distributes natural gas to industrial, commercial and residential customers in Brooklyn, Staten Island, and most of Queens. The company is the fifth largest gas distribution utility in the world, with 1,100,000 customers and annual sales of about $900 million. Ninety percent of the customers are residential households. The company employs 3,500 people, and 200 of these people work in the Information Technology area.

In 1984 the New York Regulatory Agency began requiring that Brooklyn Gas segment customers by age, and other socio-economic factors. The Regulatory Agency further required that the cost of service be tied to these factors. For Brooklyn Gas, it was no longer sufficient to only know a meter number and amount of gas usage. Specific attributes needed to be attached to each meter to determine billing rates. The existing information system was not sufficient to provide this information.

Prior to 1985, the corporate information system was an integrated mainframe running IBM VMS. The system was installed in 1971, and was so intertwined that 12 programmers were needed full-time to maintain the status quo and fix errors in the system. No improvements were being made, and the existing system could not meet the new regulatory requirements.

In 1984 Brooklyn Gas began studies for a new information system to handle all corporate information: accounting, scheduling, billing, credit checks, and marketing. In 1986, five programmers began work on a pilot version of the new system. Prototyping was done to develop user interfaces and complex routines.

It was decided that the new system should be totally separate from the old system, rather than an extension. Work on the new system began in 1987, and was completed in March 1990. The finished system had 900,000 lines of code and cost $48 million to develop. Brooklyn refers to it as the CRISII (Customer Related Information System II).

A.3.1.2 Purpose and Type of Application

The new system handles all business transactions of Brooklyn Union Gas, from meter reading to billing and maintenance scheduling. The main functions are outlined below:

- Meter reading - 1 million readings monthly input from hand-held meter readers
- Billing - 30,000 to 40,000 bills processed nightly
- Cash processing - real-time and batch processing of payments
- Credit and collection - 60,000 to 90,000 credit related transactions per day.
- Field service orders - real-time assignment of maintenance workers
- Other services - general ledger posting, statistical reports - (250 nightly)

A.3.1.3 System Size

The old system had 1.5 million lines of code. The new system has 900,000 lines of code, structured as 650 classes, with 8,600 methods. The methods contain a total of 70,000 message-sending locations.

A.3.1.4 Implementation Language

The system is written in PL/1 with object-oriented extensions to provide an execution and development environment. In 1986 the system was prototyped using Smalltalk for some of the early work.

A.3.1.5 Hardware

CRISII runs on an IBM 3090-300J mainframe under the MVS/XA operating system. About 100,000 on-line user inserts or updates are handled every day, and about 1 million insertions and deletions though batch processing are handled every night.

A.3.1.6 DBMS

Data is stored in a DB2 relational database. There are 130 relational tables, with the largest single table holding 70 million rows. Total active storage is 100 Gigabytes. An interface-level object handles all translations between the database and the object-oriented system.

A.3.2 PROJECT MANAGEMENT AND TECHNOLOGY TRANSITION

A.3.2.1 Level of Effort and Duration.

Total development time was 400 staff years spread over a 3-year period from 1987 to 1990. Peak employment was about 250 people in 1989.

The original budget for the project was $46 million. This budget was developed in 1987 after prototypes were completed, and the budget included all costs from prototyping. Actual cost was $48 million, $2 million above the estimate made three years earlier.

A.3.2.2 Staff Skills and Experience

The same team leader, Tom Morgan, was present on the project from its initial plan in 1984 until completion in 1990. Morgan led the original team that developed the Smalltalk prototype, and continued as the development project leader. Both Morgan and his direct supervisor have been Brooklyn Union Gas employees since 1971.

The project management team consisted of a core team (five key people), and eight other groups. These working groups started with 10 to 20 people each, and handled 5 major functional areas of work (meter reading, billing, credit, service processing, and cash processing). Each group assumed responsibility for the functionality within that area. In each area, this included designing the user interface, designing the interface with peripherals, and tying these peripherals into the underlying objects. Brooklyn programmers were augmented by new hires and by consultants employed by Anderson Consulting Company.

A.3.2.3 General Technology Transition strategy

This was a company-wide project discussed and planned for four years before start up. The initial concept was developed by five programmers working for six months in a separate team environment.

A.3.3 TECHNICAL APPROACH

A.3.3.1 Lifecycle Process Model

Unknown.

A.3.3.2 Standards

Unknown.

A.3.3.3 Methodologies

Custom object-oriented methods in the PL/1 programming language.

A.3.3.4 Approach for Persistent Objects

To support a transaction-oriented system, the object system assigned each user a processing thread for the duration of the user's session with the system. Each thread has an associated instance of a root Context Object. This "Context" class is the container class that holds the current object and thread.

When an object is created, it is initialized in one of two ways. A totally new persistent object is given a unique identifier, a table location, and empty attributes. A new object for an existing customer, for example, a monthly bill object, is immediately initialized with data physically stored in a relational database. Each database row has an identifier guaranteed to be unique across the 170 tables in the system.

A.3.3.5 Development environment

Tools were constructed by the team early in the project. Many traditional mainframe components were used, including IBM's ISPF, DB2, and PL/1 preprocessors.

No canned CASE tools were used on the project.

The literature indicates that the heart of the tools was an extensible entity-relation-attribute dictionary with information about the applications that makeup CRIS-II and its environment. This dictionary appears equivalent to the common data repository found in many CASE implementations.

Development was done in the IBM 3090-300 environment.

A.3.3.6 Management of Phase Transitions

The system was not implemented in phases. The system was developed incrementally and tested as each module was finished.

In June 1989, 70 MB (megabytes)of data was loaded into the new system for testing, and the first gas bill was produced by the new system. Early speed of the system was slow and required "tuning" of the object methods to improve performance. From June to December 1989, about 800 system investigations were needed each month to check on problems. In January 1990, the system was handed over to the users, and the system was stable after one week. System problems dropped to 100 per month by March 1990, and 50 per month in 1991.

A.3.3.7 Reuse of Components

In June 1990, a new hire was able to add a module of 2, 000 lines of code to the system in 3 staff-weeks of time. This module was a new interface for use by company attorneys. The module worked correctly and accessed 40,000 lines of code in the just completed CRIS-II. While this is not a demonstration of reusing the same components, it does demonstrate the productivity possible from access to existing objects.

A.3.4 PROJECT RESULTS

A.3.4.1 Current Status

The system has been in use for three years. Improvements are made as required by customers. System investigations now average 10 per month for problems or questions.

A.3.4.2 Benefits

The Information Technology office feels the new system is much improved from the previous system. There are fewer system maintenance calls, the system is faster, and 8 programmers can maintain the system instead of 12. Of more importance to Brooklyn Gas, the object orientation allows the company to adapt the system to specific marketing and billing needs.

The new system is maintained by 8 programmers; the old system was maintained by 12 programmers, a savings of $500,000 per year. The cost to develop was $48 million for 400 staff years of work. Total system size was 900,000 SLOC.

A.3.4.3 Lessons Learned

The object-oriented approach used by Brooklyn Gas did not have automatic constructor or destructor functions for objects. When an object went out of scope, the language compiler did not guarantee destruction of the object. Reclaiming storage from the heap was done with custom functions developed by the programmers. The lack of automatic constructor/destructor functions was the largest source of technical program errors. The developers noted that a real garbage collector would have been invaluable.

A.4 AMERICAN MANAGEMENT SYSTEMS, INC.

A.4.1 SYSTEM/PROJECT CHARACTERISTICS

A.4.1.1 Background

American Management Systems (AMS), Inc., is a management and information services company that recently began a transition to object-oriented technology for its information systems development. This transition was started in 1990 in response to a changing environment and customer requests for downsizing, client-server architectures, and object-oriented systems. An earlier key component of AMS's success was the development in 1980 of its "Core Foundation Software." This common layer was written in Cobol and eventually served over 600 systems. Beginning in late 1990, AMS began developing new reusable foundation software, dubbed "Core 2000 Foundation Software," that would be object oriented and take advantage of current and anticipated technologies. To date, AMS has completed five object-oriented projects and is planning to transition a large percentage of its 2,500 software developers company-wide to object and client-server technology.

A.4.1.2 Purpose and Type of Application

The type of object-oriented applications developed by AMS were information management systems such as a financial transaction management system and a facilities management system.

A.4.1.2.1 System Size

The initial object-oriented projects consisted of about 80 classes, each with 12 to 15 methods, totaling approximately 16,000 to 20,000 LOC per system. AMS is currently involved in 9 object-oriented systems, ranging from 3,000 to 8,000 function points.

A.4.1.3 Implementation Language

Smalltalk and C++ are being used as implementation languages.

A.4.1.4 Hardware

Although much of AMS's work had consisted of development on mainframes (primarily IBM), its work is moving to client/server configurations with Intel-based workstations, Unix-based servers, and host interoperability.

A.4.1.5 DBMS

AMS intends to used relational DBMS technology for the next three to five years with its object-oriented applications.

A.4.2 PROJECT MANAGEMENT AND TECHNOLOGY TRANSITION

A.4.2.1 Level of Effort and Duration

AMS's early object-oriented demonstration projects were staffed with approximately five people for about eight weeks. Its current object-oriented projects are in various stages of design and development, some with staffing levels of 20 to 30 people for approximately one year [BAE93].

A.4.2.2 Staff Skills and Experience

The existing AMS skill base was predominantly Cobol and C developers.

A.4.2.3 General Technology Transition Strategy

AMS began its transition to OOT as part of developing of its "Core 2000 Foundation Software." The use of OOT was motivated by a recognition that complex systems needed more advanced design and development techniques and methods, client requests for object technology, and by the belief that the transition from design to implementation was difficult using the more traditional, structured techniques. The AMS developers also evaluated CASE technology and information engineering tools for use in this transition. One conclusion they reached was that code generators constrained their underlying software architectures. After evaluating a number of object-oriented approaches, they determined that Jacobson's use-case approach was most effective for their analysis activities and Wirfs-Brock's method for object design.

To implement OOT, AMS's transition strategy was to "start small," develop some pilot projects, and evaluate the success of those pilots. To initiate this transition AMS chose a pilot team of five people who were recognized as "quick learners" and a system domain they already knew. This initial team received one week of classroom training in Smalltalk and object concepts, continuing with hands-on work in an eight-week apprentice program. The Smalltalk-object training has now been extended to two weeks. AMS estimates that it takes at least one month of hands-on work to understand object-oriented concepts [AMS93b]. In conjunction with object concepts, AMS developers are learning and using client-server, open system, graphics user interface (GUI), and RDBMS technology. AMS intends to train another 500 developers in 1993 and another 2,000 developers through 1995. At that point, a majority of their developers would be trained and using OOT.

A.4.3 TECHNICAL APPROACH

A.4.3.1 Lifecycle Process Model

AMS has its own lifecycle process model called "Lifecycle Productivity System" (LPS). The LPS methodology was originally based upon Structured Analysis and Design techniques for mainframe system development but which has been extended to accommodate Rapid Application Development, client-server, GUI, and object technology [BAE93].

A.4.3.2 Standards Used

The standards used are based upon client requirements.

A.4.3.3 Methodologies

AMS evaluated several object-oriented analysis and design methodologies, including those of Booch, Rumbaugh, Wirfs-Brock, Coad and Yourdon, Shlaer and Mellor, Jacobson, Martin, and Reenskaug. The conclusion of its study was that no single methodology was sufficient; in particular, "high-level analysis (domain definition) [was] not fully addressed" [AMS93a]. AMS now uses parts of different methodologies in combination. It employs the "Use-Case Driven Approach" defined by Jacobson for requirements analysis, then transitions to object-oriented design, using the Wirfs-Brock approach and CRC (Class-Responsibilities-Collaborators) for designing specific classes and roles.

A.4.3.4 Development Environment

The development environment included the Objectory tool by Objective Systems, which supports the Use-Case Driven Approach by Jacobson, Smalltalk from both ParcPlace and Digitalk, and C++ from Borland, Microsoft, and IBM.

A.4.3.5 Reuse of Components

AMS noted that the benefits from reuse may not be immediate (other than the reuse of the available base classes) since application development must be done to get reusable components. It is seeing high levels of reuse (within the 80% range) for projects where other object technology applications have already been completed.

A.4.4 RESULTS

The initial systems developed using OOT are being used for demonstration; nine new information systems are currently under development.

A.4.4.1 Benefits

Statistics regarding benefits in terms of quality, time, and cost were not available, although its research indicates that the reuse for new object-oriented projects within the same domain appears to be substantial at approximately 80%. AMS is still determining what specific metrics it should use to assess productivity gains [BAE93].

A.4.4.2 Lessons Learned

The following are some lessons learned from AMS experience with object technology [AMS93a]:

- "Object development widely misunderstood.
- Payoffs take time (1-2 years)
- Object design tools are still scarce . . .
- The 'Right' design methodology is not clear.

- Traditional CASE tools vendors are two or more years behind.
- Reuse must be accepted as a goal and planned and designed into software products.
- "New roles and responsibilities:
- Computer Human Interaction Expert
- Technical Architect
- Configuration Manager
- Object librarian (includes reuse tester)
- "Pitfalls
- Don't expect magic. Software development (especially analysis and design) is still hard.
- Insufficient training.
- Expecting significant results in the first 6 months.
- Selecting the wrong first project
- Time critical
- Too large
- Forcing the use of a structured analysis and design [when using an object-oriented language].
- Uncontrolled prototyping.
- Forgetting to think about configuration management"

A.5 SYSTEMS RESEARCH AND APPLICATIONS (SRA) CORP.

A.5.1 System/Project Characteristics

A.5.1.1 Background

System Research and Applications (SRA) Corporation is a professional technical services company that provides systems integration and development for both the government and commercial sector. SRA began considering OOT in 1986, anticipating its use for its Ada development projects. At that time, it determined that an object-oriented design approach would work better with the Ada language. Since then, SRA has used object-oriented design across a number of other applications and in other languages, including C, C++, and Smalltalk.

A.5.1.2 Purpose and Type of Applications

The SRA applications that have used OOT range from command and control, document processing, engineering data management, and image management and processing. These applications include the following major systems:

- Standard Installation/Division Personnel system (SIDPERS-3), Army WWMCCS (Worldwide Military Command and Control System) Information System (AWIS), Joint Operations Planning and Execution System (JOPES): Information management systems in Ada for personnel management and command and control.
- Picture Network International (PNI): Text and image management system in C, C++, Smalltalk.
- Storage and Retrieval for Intelligence Statistics and History (STARFISH): Document processing and management system in C++ for Joint National Intelligence Development Staff.
- Spare Parts Production and Reprocurement Support System (SPARES): engineering data management system, implemented in C++, for the Air Force Materiel Command (AFMC) at Wright-Patterson AFB, Ohio.

A.5.1.3 System Size

The Ada projects have ranged from 30 to 850,000 LOC, and the C, C++ and, Smalltalk projects have ranged from 60,000 to 200,000 LOC.

A.5.1.4 Implementation Languages

Ada, C, C++, and Smalltalk were used as implementation languages.

A.5.1.5 Hardware

The hardware is primarily Unix-based platforms, although there were some DEC VAX, Burroughs BTOS, and PC-DOS work.

A.5.1.6 DBMS

Other than the PNI system, which uses a proprietary DBMS, the object-oriented projects have used relational DBMSs such as Sybase, Oracle, and XDB.

A.5.2 PROJECT MANAGEMENT AND TECHNOLOGY TRANSITION

A.5.2.1 Level of Effort and Duration

The level varied according to specific projects.

A.5.2.2 Staff Skills and Experience

The SRA developers who learned and used OOD had a mix of skills and backgrounds, including Ada, C, and Cobol. One observation made by SRA was that the particular language background of the developers did not seem to make a difference as to whether or not someone would understand object-oriented concepts; acquiring an understanding of OOT appeared dependent upon the individual. SRA also noted that those with Cobol backgrounds were able to make the object-oriented (and Ada) transition successfully [KRI93b].

For OOT training, SRA developers received three to five days of training from David Bulman in his Model Based Object-Oriented Design (MBOOD) methodology for Ada development and some supplemental training from John Palmer in his Essential Systems Analysis.

A.5.2.3 General Technology Transition Strategy

The transition to OOT by SRA was motivated by the use of Ada, with the realization that traditional structured approaches would not work well with Ada and by the experience from its natural language/artificial intelligence group which had been using Lisp. Their first object-oriented-Ada project was on AWIS as a subcontractor to TRW. SRA transitioned to OOT for other projects, basing its decision to use OOT upon the language used and the nature of the application. Where the language was object based or object-oriented, SRA felt that object-oriented design would be the most appropriate.

A.5.3 TECHNICAL APPROACH

A.5.3.1 Lifecycle Process Model

SRA uses a lifecycle process model that is similar to that in DOD-STD-2167A but also has a business process improvement modeling phase prior to requirements analysis. In general, they employ an iterative approach to application development: first building a basic software infrastructure, then developing one thread at a time.

A.5.3.2 Standards Used

The standards used were based upon the requirements of the specific project. For most of the DoD projects, DOD-STD-2167A and DOD-STD-7935A were followed.

A.5.3.3 Methodologies Used

SRA evaluated several methodologies, including those of Booch, Berard, Buhr, Colbert, Coad, Wirfs-Brock, and Jacobson. One of its observations regarding these methodologies was that there was little separation between requirements analysis and design. While this "closeness" is often put forth as an advantage of these object-oriented approaches, SRA believed that these two activities should remain distinct, each having different objectives and, as a result, different products. The requirements specification should describe the required, externally visible behaviors of the system, while the design should describe the internal structures, functions, and algorithms. SRA expressed a concern that the specification of objects during requirements may arbitrarily impose a design and implementation constraint for constructing those objects [KRI93b].

For analysis and design, SRA employed a number of different notations and approaches. For Ada projects, SRA used Bulman's MBOOD methodology. In this approach, a domain is modeled in the analysis phase from four perspectives:

- process (data flow)
- control (state transition)
- data (entity-relationship),
- user-interface (screens)

From these models, objects and operations are identified for the subsequent design. For the non-Ada projects, SRA has used the Rumbaugh method for design and the IDEF0 and IDEF1X methods for requirements analysis. These methodologies were also supplemented with concepts from John Palmer's Essential Systems Analysis, which distinguishes between the "essential" (non-implementation dependent) and the implementation models of the system.

A.5.3.4 Development Environment

SRA has used a variety of development tools, including GE's OMT tool and ParcPlace's OBA for the PNI; OMT and Cadre's Teamwork for SPARES and STARFISH; and the Rational Development Environment for the Ada projects (AWIS, JOPES, SIDPERS). The Rational provided the capability to reverse engineer code to Booch diagrams, but not the reverse. Separate drawing tools were used to support any graphical analysis and design notations for Ada projects.

A.5.3.5 Management of Phase Transitions

SRA has not used a pure object-oriented approach on the very front end of its system developments. With the MBOOD approach, SRA first develops process, state, data, and user interface models during analysis. Then, in transitioning to design, object abstractions are derived based upon their commonality across defined processes. Those processes then serve as the operations for that object.

A.5.3.6 Reuse of Components

For SIDPERS, 41% of the code was unique to the CSUs; 10% was verbatim reuse (such as Grace and Booch components); 22% was partial reuse (from design templates);

16% was constructive reuse (resulting from code generation); and 11% consisted of Computer Software Configuration Item (CSCI) common components. Statistics for reuse within other projects were not available.

A.5.4 PROJECT RESULTS

A.5.4.1 Current Status

These projects are in various phases of development: STARFISH is in the operations and maintenance phase and the PNI system and SPARES are both in implementation.

A.5.4.2 Benefits

Although statistics were not available, SRA noted that those applications using object-oriented design seemed to require less maintenance. Particularly in the case of the Ada applications using OOT, there were fewer "withing" dependencies between packages, and hence less recompilation was required for changes.

A.5.4.3 Lessons Learned

Below is a summary of the lessons learned from SRA's experience with OOT [SRA93b]:

a. The transition to OOT may be difficult in the beginning: this transition will depend upon the specific developers, although their background may not be a factor as noted above.

b. A "clash of methods" may occur when trying to integrate old (non-object-oriented) software with object-oriented software.

c. Keep object tree/class structure shallow: SRA originally defined deep hierarchies of tightly coupled objects in its designs and subsequently found that such deep hierarchies were difficult to maintain.

d. "Regression testing becomes more crucial (particularly with C++)": one of the problems encountered was that it was not obvious which modules were affected by changes.

ACRONYM LIST

AIS	Automated Information System
AMS	American Management Systems
ANSA	Advanced Networked Systems Architecture
API	Application Program Interface
AWIS	Army WWMCS (World-Wide Military Command & Control System) Information System
BLSM	Base Level System Modernization
CASE	Computer-Aided Software Engineering
CD	Committee Draft
CDA	Central Design Activity
CD-ROM	Compact Disk - Read Only Memory
CIM	Center for Information Management
CORBA	Common Object Request Broker Architecture
COTS	Commercial off the Shelf
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DoD	Department of Defense
DBMS	Database Management System
DCE	Distributed Computing Environment
DCRM	Distributed Computing Reference Model
DDI	Director of Defense Information
DDL	Data-Definition Language
DFD	Data Flow Diagram
DID	Data Item Description
DISA	Defense Information Systems Agency
DME	Distributed Management Environment
DML	Data Manipulation Language
DODD	Department of Defense Directive
DODI	Department of Defense Instruction
EEI	External Environment Interface
FD	Functional Description
HP	Hewlitt-Packard
I-CASE	Integrated Computer-Aided Software Engineering
IDA	Institute for Defense Analyses
IM	Information Management
I/O	Input/Output
IS	Information System
ISO/IEC	International Organization for Standardization/International Electrotechnical Committee
JIAWG	Joint Integrated Avionics Working Group
JOPES	Joint Operations Planning and Execution System
LCM	Life-Cycle Management

MIS	Management Information System
NASA	National Aeronautics and Space Administration
NDI	Nondevelopmental Item
ODBMS	Object Oriented Database Management System
ODASD(IM)	Office of the Deputy Assistant Secretary of Defense for Information Management
ODP	Open Distributed Processing
OMB	Office of Management and Budget
OMG	Object Management Group
OO	Object Oriented
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OOPL	Object-Oriented Programming Language
OOPSLA	Object-Oriented Programming Languages, Systems, and Applications
OOT	Object-Oriented Technology
ORB	Object Request Broker
OSF	Open Software Foundation
OSI	Open Systems Interconnection
PM	Program Manager
RAPID	Reusable Ada Products for Information Systems Development
RDBMS	Relational Database Management System
SDD	Software Development and Documentation
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
SIDPERS	Standard Installation/Division Personnel System
SPARES	Spare Parts Production and Reprocurement Support System
SRA	Systems Research and Applications
SRI	Software Reuse Initiative
SRPO	Software Reuse Program Office
SRS	Software Requirements Specification
STARFISH	Storage and Retrieval for Intelligence Statistics and History
RFP	Request for Proposal
TAFIM	Technical Architecture Framework for Information Management
TRM	Technical Reference Model

LIST OF REFERENCES

[AMS93a]. American Management Systems, Inc., "Object Methodology, Analysis, and Design-Case Studies," Briefing, Object World '93 Conference, Boston, MA, 1993.

[AMS93b]. American Management Systems, Inc., Interviews with AMS representatives: Andrew Baer, Director of Object Technology; Gordon McDonald, and John Kemper, at DISA/CIM/XE, Fairview Park, Falls Church, VA, 25 February 1993.

[AND89]. J.A. Anderson, J. McDonald, L. Holland, and E. Scranage, "Automated Object-Oriented Requirements Analysis and Design," Proceedings of the Sixth Washington Ada Symposium, June 26-29, 1989, pp. 265 - 272.

[BAE93]. Phone interview with Andrew Baer, Director of Object Technology, American Management Systems, Inc., 28 April 1993.

[BER92]. E. Berard, Notes from "Object-Oriented Software Engineering", Course, University Extension, University of California, Irvine, California, November 1992.

[BLS93a]. BLSM Briefing notes, presented 3-4 March 1993, Maxwell AFB-Gunter Annex, AL.

[BLS93b]. BLSM Interviews, conducted 3-4 March 1993, Maxwell AFB-Gunter Annex, AL.

[BLS93c]. BLSM Questionnaire Responses, 25 January 1995, Standard Systems Center (AFCC), Maxwell AFB, Gunter Annex, AL.

[BLU91] D. Blue, *Joint Integrated Avionics Working Group (JIAWG) Domain Analysis Description*, prepared for Naval Weapons Center, China Lake, CA, by CTA Inc., Ridgecrest, CA, 30 September 1991.

[BOO83]. G. Booch, *Software Engineering with Ada,* Benjamin/Cummings, Menlo Park, California, 1983.

[BOO87]. G. Booch, *Software Engineering with Ada, Second Edition,* Benjamin/Cummings, Menlo Park, California, 1987.

[BOO91]. G. Booch, *Object-Oriented Design With Applications*, Benjamin/Cummings, Redwood City, California, 1991.

[BOO92]. Eric W. Booth and Michael E. Stark, "Designing Configurable Software: Compass Implementation Concepts," white paper, NASA/Goddard Space Flight Center, MD, 1992.

[BOO93]. G. Booch, Notes from "Introduction to Object-Oriented Design", Briefing, Object Expo 93 Conference, New York, N.Y., April, 1993.

[BRO82]. F. Brooks, *The Mythical Man-Month*, Addison-Wesley, Reading, MA, 1982.

[BUH90]. R.J.A. Buhr, *Practical Visual Techniques in System Design With Applications to Ada,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[CAT91]. Roderic Geoffrey Galton Cattell, *Object Data Management: Object-Oriented and Extended Relational Database Systems,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

[CHA92]. D. de Champeaux, A. Anderson, & E. Feldhousen, "Case Study of Object-Oriented Software Development," OOPSLA92.

[CHI90]. E, J. Chikofsky and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990.

[COA90]. P. Coad and E. Yourdon, *OOA -- Object-Oriented Analysis*, 2nd Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[COX86]. B.J. Cox, *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, Massachusetts, 1986.

[DAH70]. O. Dahl, B. Myhrhaug, and K. Nygaard, *The Simula 67 Common Base Language*, Publication No. S-22, Norwegian Computing Center, Oslo, 1970.

[DAV93]. J. Davis and T. Morgan, "Object-Oriented Development at Brooklyn Union Gas," *IEEE Software*, January 1993.

[DIJ69]. E. Dijkstra, Notes on Structured Programming, *Structured Programming,* Academic Press, 1969

[DIS91]. D. Diskin, "Guidelines and Counting Rules for Contel's Software Metrics", Technical Note CTC-TN-91-001, GTE, Waltham, Massachusetts, 1991.

[DIS93]. D. Diskin, Notes Regarding DoD Staffing at Central Design Activities, March 1993.

[DMR92]. DoD, DMRD 918. Defense Management Review Decision - 918, Subject: Defense Information Infrastructure, 1992.

[DOD83]. Department of Defense, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A-1983, 17 February 1983.

[DOD91]. *Department of Defense Software Technology Strategy*, Draft, prepared for the Director of Defense Research and Engineering (DDR&E), December 1991.

[DOD92] Department of Defense Reuse Program Initiative Office, "DoD Software Reuse Vision and Strategy," *Crosstalk,* No. 37, October 1992.

[DOD92a] Department of Defense, Data Administration Strategic Plcan, Defense Information Systems Agency, July 1992.

[EMB92]. David W. Embley, Barry D. Kurtz and Scott N. Woodfield, *Object-Oriented Systems Analysis: A Model-Driven Approach,* Yourdon Press, Englewood Cliffs, New Jersey, 1992.

[FAY91] M. Fayad et al., "Mission Generation System (MGS): An Application of Shlaer-Mellor's Object-Oriented Method," *IEEE/AIAA 10th DASC '91 Proceedings*, IEEE Press, 1991.

[FIC92]. R. Fichman and C. Kemerer, "Adoption of Software Engineering Process Innovations: The Case of Object-Orientation," Sloan WP No. 3434-92, Sloan School of Management, Massachusetts Institute of Technology, IEE Article, June 1992.

[FIC92a]. R. Fichman and C. Kemerer, "Object-Oriented and Conventional Analysis and Design Methodologies," *IEEE Computer*, Vol. 25, No. 10, October 1992.

[FIR92]. D. Firesmith, "Take a Flying Leap: The Plunge into Object Oriented Technology", *American Programmer*, Vol. 5, No. 8, October 1992.

[HAR92]. "Lessons Learned (LL) for the Base Level System Modernization (BLSM)", prepared by Harris Data Services Corp., Standard Systems Center (AFCC), Maxwell AFB, Gunter Annex, AL. 14 September 92.

[HAR93]. "Software Development Plan (SDP) for the Base Level System Modernization", Harris Data Services Corp., Standard Systems Center (AFCC), Maxwell AFB, Gunter Annex, AL, 15 January 93.

[ICA92] I-CASE Program Management Office, "Integrated-Computer Aided Software Engineering (I-CASE) Request for Proposal," Standard Systems Center, Gunter AFB, AL, 1992.

[JAC92]. Ivar Jacobson, Magnus Christerson, Patrik Jonsson and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

[JAC93]. I. Jacobson, "Is Object Technology Software's Industrial Platform?," *IEEE Software,* January 1993.

[KEM91]. C. Kemerer and S. Chibambeer, "Towards a Metrics Suite for Object-Oriented Design", Proceedings of OOPSLA 1991, ACM Press, 1991.

[KIM90]. Won Kim, *Introduction to Object-Oriented Databases,* MIT Press, Cambridge, Massachusetts, 1990.

[KOR93]. T. Korson, "A Research Grant Proposal to DoD", 8 March 1993.

[KRI93a]. Interview with David Kriegman, Director, Software Technologies, Systems Research and Applications Corporation, 10 March 1993.

[KRI93b]. Phone interview with David Kriegman, Director, Software Technologies, Systems Research and Applications Corporation, 20 April 1993.

[KRU92] C. Krueger, "Software Reuse," *ACM Computing Surveys*, Vol, 24, No. 2, June 1992.

[LEA90] B. Leathers, "Panel: OOP in the Real World," *Proceedings of the European Conference on Object-Oriented Programming Languages ECOOP '90,* Ottawa, Canada, October 1990.

[LEE93]. C. Lee and S. McClure, "The Migration of the Cobol MIS Community to Object Technology - A White Paper", SEMAPHORE, 1993.

[LEI93]. E. Leinfuss, "Managing Class Libraries Takes Discipline," *Software Magazine, Client/Server Computing Special Edition*, January 1993.

[LEW91]. J. Lewis, S. Henry, D. Kefura, & R. Schulman, "An Empirical Study of the Object-Oriented Paradigm and Software Reuse," OOPSA91.

[LIS74]. B. Liskov and S. Zilles, "Programming with Abstract Data Types," *Proceedings of ACM SIGPLAN Conference on Very High Level Languages,* ACM 1974.

[LOR93]. M. Lorenz, *Object-Oriented Software Development - A Practical Guide*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[MAR92]. James Martin and James Odell, *Object-Oriented Analysis & Design,* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[MEY88]. Bertrand Meyer, *Object-oriented Software Construction,* Prentice Hall, Englewood Cliffs, NJ, 1988.

[MIT87]. MIT, *Object-Oriented Concurrent Programming*, MIT Press, Cambridge, Massachusetts, 1987.

[MAZ92]. L. Mazzucchelli, "Holding onto what really matters," *Object Magazine*, November-December 1992.

[MEY88]. B. Meyer, *Object-Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[NEL91]. M. Nelson, "An Object-Oriented Tower of Babel," *OOPS Messenger*, ACM Press, 2,3 July 1991.

[NIE89]. O. Nierstrasz, "A Survey of Object-Oriented Concepts," *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, New York, 1989.

[PAG92]. M. Page-Jones, "Object orientation: the importance of being earnest," *Object Magazine*, July-August 1992.

[PAR72]. D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, 15, May 1972.

[PAR92]. D. Parkhill, "Object-oriented technology transfer: techniques and guidelines for a smooth transition," *Object Magazine,* May-June 1992.

[PIT93]. M. Pittman, "Lessons Learned in Managing Object-Oriented Development," *IEEE Software,* January 1993.

[PFL89]. S. Pfleeger, "Recommendations for an Initial Set of Software Metrics", Technical Report CTC-TR-89-017, GTE, Waltham, Massachusetts, 1989.

[PFL90]. S. Pfleeger and J. Palmer, "Software Estimation for Object-Oriented Systems", *Fall International Function Point Users Group Conference*, October 1990.

[PRI87] R. Prieto-Diaz and P. Freeman, "Classifying Software for Reusability," *IEEE Software*,Vol. 4, No. 1, January 1987.

[RUM91]. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[SHA93]. R. Sharble and S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", *ACM Software Engineering Notes*, Vol. 18, No. 2, April 1993.

[SRA93a]. Systems Research and Applications Corp., "SRA Corporation Model Based Object-Oriented Design," Briefing at IDA, 10 March 1993.

[SRA93b]. Systems Research and Applications Corp., "SRA Corporation Experience with Object-Oriented Technologies," Briefing at IDA, 10 March 1993.

[SHL88]. S. Shlaer and S.J. Mellor, *Object-Oriented Systems Analysis: Modeling the World In Data*, Yourdon Press: Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[SHL92]. Salley Shlaer and Stephen J. Mellor, *Object Life-styles: Modeling the World in States*, Yourdon Press, Englewood Cliffs, New Jersey, 1992.

[SNY93]. A. Snyder, "The Essence of Objects: Concepts and Terms," *IEEE Software,* January 1993.

[STA93a]. Mike Stark, "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies," Goddard Space Flight Center, MD, 1993.

[STA93b]. Interview with Mike Stark, engineer at NASA/Goddard Space Flight Center, Computer Sciences Corp., Greenbelt, Maryland., 29 March 1993.

[STA87]. M. Stark and E.V. Seidewitz, "Towards a General Object-Oriented Ada Life-Cycle," *Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium*, U.S. Army Communications-Electronics Command, Fort Monmouth, New Jersey, pp. 213 - 222.

[STR91]. P. Strassman, Briefing on Corporate Information Management Initiative, 1991.

[TAF92]. DoD, Technical Architecture Framework for Information Management, 1992..

[TAL93] Taligent., "Lessons Learned from Early Adopters of Object Technology," white paper, Taligent, Inc., Cupertino, CA, 1993.

[TAY90]. David A. Taylor, *Object-Oriented Technology: A Manager's Guide,* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1990.

[TAY92]. David A. Taylor, *Object-Oriented Information Systems: Planning and Implementation,* John Wiley & Sons, Inc., New York, New York, 1992.

[TAY93]. D. Taylor, “Software Metrics for Object Technology”, *Object Magazine*, March-April 1993.

[YOU89]. E. Yourdon, *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[WAL93a]. Sharon Waligora and James Langston, “Maximizing Reuse: Applying Common Sense and Discipline,” white paper, Computer Sciences Corporation, MD, 1993.

[WEG90]. P. Wegner, “Concepts and Paradigms of Object-Oriented Programming,” *OOPS Messenger,* ACM Press, 1,1, August 1990.

[WIL90]. F. Wild, “A Comparison of Experiences with the Maintenance of Object-Oriented Systems: Ada vs. C++”, Proceedings of Tri-Ada ‘90, Baltimore, Md., 1990.

[WIL93]. N. Wilde and P. Matthews, “Maintaining Object-Oriented Software,” *IEEE Software,* January 1993.

[WIR90]. R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.